RATIONAL AGENTS AND THE PROCESSES AND STATES OF NEGOTIATION

SHAMIMABI PAUROBALLY

BEIT SCIENTIFIC RESEARCH FELLOW

A thesis submitted for the degree of
Doctor of Philosophy of the University of London and for the
Diploma of Membership of the Imperial College,
September 2002

Department of Computing
Imperial College London

Abstract

This thesis shows how a verified and unambiguous theory of a protocol with known properties enables rational agents to interact in a negotiation process and to finally satisfy their goals using strategies and plans. This is achieved through an application of an extended form of propositional dynamic logic in the verification, validation and reasoning about interaction protocols in a multi-agent system.

Agent interaction, as a key aspect in multi-agent systems and automated negotiation, has lead to a number of proposed agent communication languages and protocols. In contrast to a language, a rational agent can reason about a protocol to strategically plan possible courses of action in a bid to achieve its goals. Existing techniques for specifying protocols have resulted in faulty and ambiguous interaction protocols, leading to contradictory beliefs between agents. There remains a need for formally specifying and validating sharable interaction protocols with desirable properties. This thesis specifies, verifies and analyses protocols for automated negotiation through the application of Artificial Intelligence techniques.

To this end, a meta-language called ANML is specified as an extension of propositional dynamic logic. The syntax and the semantics of ANML, including axioms and inference rules that hold in this normal modal system, are defined. Interaction protocols between agents can be concisely and completely specified in ANML allowing representation and reasoning about the states and processes of a negotiation. Examples of protocols for various types of negotiation are given as logical theories in ANML.

This thesis verifies interaction protocols proposed in statecharts and AUML, [Odell and al. 2000], and show the inadequacy of these notations for specifying communication in multi-agent systems. For each verified protocol, its correct version is given in ANML. In addition to correctness, a protocol may exhibit safety, liveness or game theoretic properties. Axioms in ANML for a range of safety and liveness

properties in an interaction protocol are defined. As an example, these properties are proved for a given bilateral protocol. This methodology can be used for designing or choosing between protocols with required properties.

Another benefit of the ANML meta-language allows the problem of ensuring consistency between different participant's beliefs to be addressed. This thesis provides and proves the conditions for preventing contradictory beliefs and knowledge between agents about a negotiation protocol and state – even in an imperfect communication medium.

Finally bilateral protocols and strategies for evaluating and generating sets of issues negotiated over are combined as a vehicle for simulation. The performance of each strategy is analysed from test results. Paths towards a goal state may be derived from such protocols and strategies, engendering a planning capacity in an agent.

ANML is found to be expressive enough to specify dynamic *m*-*n* agent interactions. Further developments are possible for applying our methodology in a wider scope of agent interaction. In addition, the expressiveness of ANML could be increased and the properties of the formalism analysed.

Acknowledgements

First of all, I thank my supervisor Jim Cunningham for the inspirational ideas in this thesis and his support throughout my postgraduate years at Imperial College. His understanding and witticism have constituted an encouragement for me. I also thank the examiners Frank Dignum and Stefan Poslad for their feedback to improve the thesis. Many thanks to my second supervisor Keith Clark, my colleagues in the Communicating Agents Group and in the Computing Department at Imperial College. In particular, I have appreciated the interesting discussions with Lloyd Kamara, Jeremy Forth, Iain Stewart, Alexander Yip, Miguel Leith and Athanassios Diacakis. I would like to also thank Nick Jennings and Phillip Turner at Southampton University. Thanks Phillip for helping me during the last few days before submission. I spare a thought for the friends that I have acquired throughout my studies in UK. I send my special regards to Yasseen for his endless support and encouragement to keep on working.

I thank the Beit Fellowship and the ORS Committee for the financial funding during three years of this Ph.D. I acknowledge the support of the OSM project during the first year. This project has introduced me to the area of electronic commerce and negotiation.

Most of all, I would like to thank my parents and my family in Mauritius for their love and care during all my life. My mother and father have always patiently awaited my visits and have never ceased to be a source of inspiration. I also thank Sayed Saheb for always being there for me and for attending my birthdays. Undoubtedly, this thesis and any accomplishments would not have been possible without the help from Allah and his messenger Muhammad (peace be upon him).

Contents

Figures

1 Introduction

Latin proverb: '*quid pro quo. clara pacta, boni amici'*.
'*Something for something. Clear agreements, good friends.*'

1.1 Negotiation in Agent Mediated Electronic Commerce

Information and communication technologies facilitate online trading by allowing participants to deal in commercial transactions at reduced costs, regardless of geographical constraints, and to access virtual markets containing up-to-date information 24 hours a day. There has been substantial research in security and payment systems, [Wang and al. 1998], allowing most online trading sites to provide encrypted and secure electronic payment options. However, current electronic commerce applications do not exploit the full potential of the Internet. Electronic commerce may be defined as performing commercial transactions via the Internet, which involves more than just shopping malls. Most online shopping sites belong to the first generation of e-commerce applications with client and server software. Humans are still involved at either end, adding to transaction costs. Once the issues of user acceptance and trust of online trading systems are resolved, there is scope for complex interactive online transactions. Such interaction could be in the form of automated negotiation, matching suppliers' services to customers' requirements, electronic shopping assistants, aggregation of services from various sources, automated authentication to establish credentials and automated logistic systems to ensure supply and delivery.

A tenet of this thesis is that for electronic commerce to proliferate at this higher level, there must be an open architecture that allows participants to communicate and coordinate irrespective of differences in location and platform. Several distributed object architectures have been proposed for satisfying the interactive aspects of electronic commerce, [OSM 1997]. Although there is considerable industrial interest in the development of interoperable distributed object-oriented software, some concern has arisen amongst academic software engineers over the inherent complexity of systems built this way, [Martin and al. 1999b; Webster 1995]. Higher level architectures on top of object frameworks begin to come in the realm of agent oriented systems, which are at a natural higher level of abstraction and inherit from object frameworks to allow automation of online services.

Agent research is one of the fastest growing fields in Information Technology. An agent can briefly be described as a processing entity which, when situated in an environment, senses the environment and acts autonomously upon it. It can use stored knowledge to act without the intervention of its owner in a dynamic environment, responding to changes in the environment, interacting with other agents and taking initiative where appropriate. Software agents that are personalised and continuously running may be used to reduce costs and to automate time-consuming tasks. For example, in online auctions extending over weeks, such as Ebay, [Ebay 2002], an agent knowing the preferences of its owner may act as an auctioneer, a seller or watch out for interesting auctions and strategically bid no further than its reserve price. It can take on several roles and participate in multiple transactions or be designed towards one task. One or more specialised agents can support different steps during a transaction, from searching and filtering large unstructured online information to brokering, negotiation and payment. New software architectures based on autonomous, "intelligent" software agents promise elegant and evolutionary solutions to electronic commerce, but they have yet to be proven.

Negotiation is an important aspect of commerce, [Cunningham, Paurobally and al. 1998]. It is a form of decision-making where two or more parties jointly search a space of possible solutions with the goal of reaching a consensus. Negotiation is part of those wider dynamic processes whereby commercial goals are achieved by the parties to a contract. Overt negotiation, as deal making, is often suppressed by agreed

rules of encounter, but it is rarely absent altogether. Examples of negotiation in real life can be observed in daily conversations, auction houses and commerce. A protagonist does not have to decide on an exact offer and can negotiate a dynamic deal that takes into account features other than price. In electronic negotiation, participants interact for mutual gain through an electronic commerce framework. However all parties have to be physically present in a manned electronic negotiation, which may extend over a long period of time, leading to increased transaction costs. Online auctions such as Ebay and Amazon remove the constraint of all participants being co-located but still require users to manage their negotiation. Automated negotiation has the potential to save both time and management costs. There are at least anecdotal reports that electronic agents may be able to find better deals in strategically complex environments without the drawbacks of human ego and prejudices, [Paurobally and Cunningham 2002]. Software agents can participate in parallel and complex negotiations and monitor these over long time periods, which would be difficult for a single human. Automated negotiation is perhaps the most fundamental and powerful mechanism for managing inter-agent dependencies at run-time, [Jennings and al 2000]. It is useful in both cooperative and competitive situations for achieving an agent's goal.

1.2 Motivation and Aims

A motivation for this thesis is to help electronic commerce flourish by expediting negotiation. To do so, a framework is provided with a meta-language for representing and reasoning about negotiation, for its automation or partial automation as a step towards agreement resolution. The meta-language is an extension of propositional dynamic logic and is used to formally specify and reason about multi-agent protocols for negotiation.

Research in automated negotiation has gained impetus in the last decade. Inexpensive communication infrastructure and the rise of virtual enterprises favour negotiation. The contract-net protocol, [Smith 1980], is one of the first instances of protocols for electronic negotiation. Past applications were quite basic, typically focusing on a single issue and on price. Current forms of commercial online negotiation concentrate on auctions where the parameters and the domains are well defined. Examples of

online auctions are QXL, [QXL 2002], Ebay, [Ebay 2002], Amazon, [Amazon 2002], Last-minute, [Last-minute 2002], and travel companies such as Lufthansa and Nouvelles-frontieres, [Nouvelles-frontieres 2002]. More advanced negotiation is possible involving greater flexibility in interaction between participants. Different users may have different preferences and many are interested in not only price but also other value-added aspects of a purchase, such as quality and delivery. Applications need practical agents that negotiate over a variety of issues, find an agreement that satisfies all the parties, and prepare contracts on behalf of their owners. See [Chavez and Maes 1996; Sandholm 1999; Wurman 1998] for research on various types of auctions, applications and strategies and [Kraus and al. 1998; Rosenschein and Zlotkin 1998; Sierra and al. 1998] for argumentation and game theoretic techniques.

A key aspect in real or automated negotiation is the rich, often implicit, interaction between (human or agent) parties, using languages and protocols. Setting up a negotiation involves deciding on a common language, ontology and choosing a common protocol. A common language establishes acts for communication that are known by all participants. A common ontology ensures messages have the same meaning for all of the agents in that context. In real life, people communicate verbally with a common language and share a common ontology to avoid misunderstandings. The same requirements apply to interaction in multi-agent systems where agent communication languages may be used, but deciding on a common ontology is a major problem in its own, [Huhns and Singh 1997].

Having a standardised communication language removes the need for agents to design a common language for interaction but there have been some criticisms about the lack of formalisation and ambiguous nature of existing agent communication languages, [Labrou 2001; Pitt and Mamdani 1999]. As such it is probable that more than one agent communication language, (ACL), will emerge. Among the number of agent communication languages proposed in the last decade, KQML, [Finin and al. 1993], and FIPA ACL, [FIPA 1997], both based on speech act theory [Austin 1962; Searle 1969], are the most prevalent and commonly used. FIPA agent communication language, (FIPA ACL), emerged after criticisms about the lack of formalisation in KQML and included the semantic language, SL, as a formal language for defining the

semantics of FIPA ACL. However, SL itself has to be precisely defined and its intentional semantics are overly complex. There remain several issues that need to be resolved in KQML and FIPA ACL for practical use, [Labrou 2001; Pitt and Mamdani 1999; Wooldridge 1998]. Some of these issues are low-level, for example, interoperation between agent systems for the success of agent deployment on the Internet. Other issues concern registration and facilitation primitives. The theory behind FIPA ACL and KQML are based on propositional attitudes, but both have the disadvantage of allowing ambiguities in the mental states of an agent. The choice and meaning of performatives are still unclear and KQML and FIPA ACL do not cover the exchange of more complex contents such as shared plans, goals or strategies. These issues are important because to be adopted for standardisation, an ACL needs to consider both the theoretical and the pragmatic approaches.

The embedded content in an agent's message is expressed using an ontology, corresponding to a specific domain, and a content language. Typically an agent will know one or more ontologies. A logic content language can be used to specify an agent's mental model of the world (e.g. beliefs, desires, intentions). KIF, (Knowledge Interchange Format), is a language proposed in the KQML specification to support content languages in ACL messages. FIPA ACL does not commit to a particular content language. Content languages for knowledge representation for WWW content include DAML+OIL, [DAML] and RuleML, [RuleML].

Another requirement for negotiation is a common protocol. Agents often do more than send a single message in a conversation. Rather they engage in task-oriented, shared sequences of messages to which all of them comply, in order to achieve specific tasks, such as a negotiation. Conversations and protocols are a shift from individual messages to sequences of messages. When an agent sends a message, it can expect a receiver's response to be among a set of messages indicated by the conversation structure and history. These expectations derive from a higher-level structure such as a *protocol*. A protocol or *conversation policy* defines a sequence of message exchanges that agents engage in order to perform certain tasks. Two or more agents agree to follow this sequence in communication with one another. A conversation lends context to the messages exchanged and facilitates interpretation of the meaning of a message, [Labrou 2001]. A protocol can be regarded as public rules

or a norm advocating the conduct of an agent towards other agents when carrying out some negotiation. A common protocol ensures that all participants following it will coordinate meaningfully and can expect certain responses from others. A protocol can be application or domain specific, but as long as all the participants know and follow it, a conversation can be carried out, which may eventually end up in a terminal state. The protocol to be complied with may be pre-arranged upon entering a negotiation and in some cases can be considered common knowledge. Flexible, non-fixed protocols are also possible. Common knowledge of a protocol in a group means that every agent knows the protocol and everyone knows that everyone knows the protocol and so forth. One way to ensure that all agents learn about a protocol and its meaning is to have a public repository of protocols and their semantics. There are a number of well-known protocols that can be used in electronic negotiation such as English, Dutch and sealed-bid auctions. The KQML specification suggested an implicit sequencing of messages in agent interactions. Both protocols and ACLs aim to facilitate agent communication. A protocol guides how messages from a library of ACLs are sequenced. Some researchers have taken a protocol-oriented approach towards ACL semantics in order to standardise agent communication.

However there are cases where existing languages, domain ontologies and protocols are not suitable for a negotiation. It is inefficient or unfeasible to derive new protocols and languages for all imaginable cases and the participants themselves may not have the expertise to do so. Although protocols have become part of many multi-agent infrastructures, there is still a lack of methodologies for formalised protocol specification and implementation. Traditionally, deterministic finite state machines have been used to specify protocols. Early work on implementing and expressing conversations can be found in [Winograd and Flores 1986]. The COOL system, [Barbuceanu and Fox 1995], describes agent conversations through detailed finite state machine models, but two different agents must use different automata to engage in the same conversation. Other research on protocols using finite state machines are found in [Kuwabara and al. 1995] (Agentalk), [Wagner and al. 1999] and [Elio and Haddadi 1999] (COSY). [Nodine and Unruh 1997] (InfoSleuth) and [Pitt and Mamdani 1999] use deterministic finite state automata to specify protocols for BDI agents. Other approaches for developing protocols include Petri nets, [Cost and al.

1999], state charts, [Moore 1999], Dooley Graphs, [Parunak1996] and AUML, [Odell and al. 2000].

Most of these methodologies are diagrammatic notations that are unsuitable for fully expressing multi-agent interactions. Protocols expressed in them are prone to errors and ambiguities, as shown in chapters 4 and 5. Thus more research is needed on the formal aspect of specifying protocols to facilitate the sharing and use of conversation policies by agents. Sophisticated and unambiguous interactions need formalisms with more support for concurrency and verification, defining the roles of involved agents along with the constraints between messages. This thesis aims to fulfil this requirement for tools for conversation specification, verification and sharing. The main objective of this thesis is to show that our framework can be used for specifying, verifying and reasoning about protocols with desirable properties in a negotiation involving rational and goal-oriented agents. The thesis answers the following need: firstly how can protocols be best defined for agents to understand them and each other without the risk of inconsistency between their mental states. Secondly how can an agent use a specification standard to define the protocols it is willing to engage in and to learn about other agents' protocols and their properties. Thirdly, an approach of how to extend existing protocols through creation of new and more detailed versions or by combining protocols in a new compound protocol.

As part of a set of specification tools, the ANML meta-language is proposed which can be used to build and reason about protocols and provide an abstract theory of a conversation. The meta-language is based on propositional dynamic logic so as to provide intuitive theories for negotiation. ANML – Agent Negotiation Meta-Language – supports the concise and full specification of protocols such that they can be shared and they encapsulate desirable and testable properties. Well-defined protocols expressed in ANML can be used to coordinate agents in their goal seeking activities. Such coordination entails interaction with specific properties. Communicatives in existing agent languages may be passed as parameters in ANML. An ANML protocol specifies the structure of and sequence in a conversation, while individual messages can be in ACLs like KQML or FIPA ACL. As an analogy to real life conversations, ANML may be considered as a tool to construct and verify the

grammar in a conversation where the words being communicated are in English (or ACL).

1.2.1 Contributions to Automated Negotiation

This thesis aims to satisfy the need for a formal specification notation for representing protocols and checking their properties. It is argued that ANML concisely specifies protocols in a more formal and precise way than finite state machines or AUML and easily supports the representation of constraints. An abstract analysis to define axioms for a number of properties is provided and it is proved how an example protocol in ANML can exhibit correctness, safety and liveness properties. The existence of a joint theory between negotiating agents is assumed, accompanied with a discussion of how to maintain its consistency. The contribution of this thesis to the field of automated negotiation is fivefold as follows:

The *first contribution* lies in a meta-language (ANML – Agent Negotiation Meta-Language) to represent and reason about the states and processes of negotiation. It is difficult to find a notation where processes and states are given equal status, let alone form the basis for a simple and rational calculus for an executable system. Traditional imperative programming languages, such a C and Pascal leave reasoning about programs to external axiomatisations which impose complexity and are prone to incompleteness, [Waldinger and Levitt 1974]. Object oriented languages provide otherwise missing capacity for data abstraction, but without solving the problems of the need for a rational calculus. Dynamic logic, [Pratt 1976], is rare in providing reasoning about the effect of processes on states of affairs, but in its primitive form it lacks process abstraction, and has no seriously executable form as a programming system. Thus there is no established notation to even represent both the states and processes of an active agent, let alone a calculus for deciding why a particular negotiation should achieve the mutual goals. The application of ANML in specifying a protocol yields an intuitive logical theory of a negotiation in terms of its states and processes. The syntax of ANML is given in a program logic similar to that used in [Goldblatt 1987] and is an extension of propositional dynamic logic. A *meta-language* is defined as a language in which to discuss the truth of statements in another language. Communicative acts in an agent communication language can be passed in ANML and the state of common and joint beliefs in a group of agents

analysed. This meta-reasoning capability justifies why ANML is a *meta-language*. The semantics of ANML constructs are defined using Kripke structures, [Chellas 1980], together with an axiomatisation of ANML as a normal system. One important notion that arises is the *state of a negotiation*. The state of a negotiation exists by virtue of being shared belief in a group of agents engaged in that negotiation process.

The *second contribution* consists of building interaction protocols and verifying protocols proposed in finite state machines, statecharts and AUML. A protocol is expressed as a logical theory with axioms for relations between states and processes. ANML provides a sufficient level of abstraction for reuse of protocol fragments in various types of negotiation. This logical approach is especially suitable for verification purposes. Ambiguities and errors may be discovered and removed by expressing existing and new protocols in ANML. This is an important benefit, since a negotiation may become inconsistent or reach an undefined state because its protocol is misunderstood and ambiguous, thereby causing conflicts between participants. ANML ensures that any ambiguities are ironed out to yield complete and clear sharable theories as protocols. This thesis shows that statecharts and AUML protocols contain errors and are unsuitable for expressing multi-agent interactions. ANML has enough expressiveness for specifying *m-n* agent interactions. For each verified protocol, its corrected version is given in ANML and when possible in their original notation.

A major problem in agent communication is obtaining a common ontology for all parties. A shared ontology implies that a group of agents share the same semantics for what they are communicating about in a given context. Even in real life conversations where humans have inherent knowledge about languages and cultures, misunderstandings arise because of different ontologies. A compromise solution involves establishing *domain ontologies*, [Farquar and al. 1996]. Likewise, if two agents have different protocols, there must be a way to establish an equivalence between these protocols or to derive or extend them to a common protocol. ANML is a step towards finding similarities between individual protocol theories for a common protocol. Techniques such as theorem proving may be used for deriving a common theory from logical representations of individual protocols in ANML.

The *third contribution* lies in determining the properties of a protocol or extending it to exhibit required properties. Because ANML yields the logical theory of a protocol, this sets a precedent as it enables us to analyse the properties of that protocol. Protocols may be chosen according to their properties. This thesis specifies axioms for safety, liveness and related properties for a protocol in ANML. Compared to temporal logics systems, our framework provides a relatively easy method for checking properties such as termination, safety, completeness and liveness in a protocol.

The *fourth contribution* is the capability for ensuring that the joint knowledge and beliefs in a group about a negotiation remains consistent. For example, consider two agents negotiating according to a common protocol expressed in ANML. An agent can customise the common protocol privately for its purposes. Each agent has their individual knowledge and beliefs for that negotiation and there is a joint mental state, which is the combination of their individual mental states. Agents realistically do not know the joint knowledge and so inconsistency may arise in the joint knowledge even though their individual knowledge are consistent on their own. The same requirements apply for consistent joint beliefs in a group about a negotiation state. For example, one agent may believe that the state of negotiation is *agreed* while another believes that the state is *requested*. Even though their individual beliefs are consistent, the consequent of the union of their private beliefs is inconsistent leading to an overall inconsistent negotiation. This thesis gives the conditions for respectively maintaining and attaining consistency between the individual knowledge and beliefs of a group of agents, where their individual knowledge are extensions of a common knowledge. This thesis also provides rules and assumptions for synchronous state transitions in the beliefs of a group of agents in an imperfect communication medium.

The *fifth contribution* is the planning capability of agents using ANML. Contrary to ACLs, protocols allow for the planning of sequences of actions. ANML fits naturally with dynamic logics of action and multi-modal logics to represent the states and processes of a negotiation. Possible paths, consisting of actions or sub-processes, leading to a goal state may be derived from the theory of a protocol. A reasoning system is thereby obtained, which can relate processes to goal states. This engenders a planning capability in an agent for deriving a partially ordered set of possible paths

to a goal. An agent may use decision making mechanisms for one step evaluation and generation of a set of issues, as in [Faratin and al. 1999], and planning mechanisms such as path-finding, pruning out the worst paths and choosing between possible paths for sequence of actions.

As an analogy consider real world interactions, where Jacques, from France, and Julio, from Spain, who both speak their local language and English, want to converse. To understand each other, they need a suite of established methodologies amongst which are a common language, protocol and ontology and their semantics. Jacques and Julio's common language is English. If they did not have a common language then they could use gestures but this is difficult to translate to the agent world and represents another problem. A common language between two agents could be FIPA or KQML. If Jacques says "hello", Julio may reply with a similar greeting like "hello" and when Jacques asks a question, then Julio might answer that question. It would be erroneous if Jacques says "hello" and Julio responds with "yes, I accept your offer for a car". Jacques did not ask a question and there is also an issue of semantics and ontology involved in the conversation. They both follow an implicit protocol of conversation, deriving from their cultural knowledge, in order to share or achieve something. We specify this protocol explicitly in the world of agents. The contribution of this thesis lies in a methodology to build that protocol and obtain an intuitive theory of their interaction. In addition, ANML may be used for verifying that the theory (i.e. protocol) is error-free and clear, irrespective of the language used. There should be no misunderstandings between Jacques and Julio on the responses to a valid message and how to conduct a conversation. Therefore, agents Jacques and Julio are supplied with a framework for building an unambiguous protocol with verified properties and for reasoning about the states and processes of their conversation in order to plan for a goal, for example an agreement. In effect, Jacques and Julio are given a tool for constructing and verifying the grammar of their conversation in any language of their choice. Since they are from different cultures, Jacques and Julio may have different ways of interacting with people. They can agree on a norm that they both understand by specifying the structure of sentences so as to achieve a meaningful conversation for mutual gain. A common theory can be derived from their individual knowledge as ANML is independent of culture and language differences.

This thesis describes how an agent can build and verify a theory for a protocol and its properties and thus a negotiation and how it can use this theory to reason about the states and processes of negotiation towards planning for achieving its goal. These facilities are offered through the specification of a meta-language allowing an agent to derive an abstract theory of negotiation.

1.3 Thesis Outline

Chapter 2 situates this thesis in the field of agent research. It contains a critical analysis of various agent-oriented concepts and systems, for example logic-based, multi agent systems, intentional and reactive architectures. It includes a survey of the state of the art in agent mediated electronic commerce, agent negotiation and its requirements.

Chapter 3 justifies the need for ANML for constructing theories of a negotiation, with respect to existing methodologies. It surveys possible logics for specifying ANML. The meta-language derives from dynamic and multi-modal logic and inherits their axioms and inference rules. The semantics of ANML is specified both informally and formally using standard models for possible worlds. In giving the axioms and standard proof rules for a normal ANML modal system, it is shown that the *S5* properties do not apply in ANML.

Chapter 4 presents applications of ANML using explanatory scenarios. Protocols between groups of agents are expressed as statecharts and in ANML. The example protocols covered include those for negotiation between many parties through raising and voting on motions, for making promises and calling on them, for different forms of bilateral protocol and auction protocols.

Chapter 5 discusses and illustrates protocol verification. The first case study is that of negotiation between two agents (a bilateral negotiation), describing how errors and ambiguities can be found and corrected in the original state transition diagram. Additional case studies include various interaction protocols in AUML proposed by FIPA. Corrected versions of all the verified protocols are specified in ANML.

Having obtained a logical theory of a protocol in ANML, the properties of that protocol can be derived. Chapter 6 defines properties in ANML for liveness, termination, safety and soundness, serialisability, complexity, security, reliability, consistency, decidability and fairness. The bilateral protocol is used as an example to prove such properties. The issue of deciding on the equivalency between protocols (in a bid to share, extend or derive a common protocol) is discussed.

Chapter 7 addresses the problem of maintaining the joint consistency of a group's knowledge and beliefs. To solve this problem, possible formal properties of a theory such as completeness, consistency, satisfiability are defined. Conditions that a common theory (protocol) and all agents' individual knowledge have to satisfy are declared to ensure that the joint knowledge remains consistent. This chapter also addresses the problem of attaining consistency in a group's joint beliefs about a negotiation state, even when the underlying communication medium is imperfect. In order to solve this problem at the communication level, assumptions and rules are provided for ensuring synchronisation in belief revision about the negotiation state.

Chapter 8 considers strategies as addition to protocols for rational decision making. Strategies for decision making without looking ahead, called one-step strategies, can be found in [Matos and al. 1998]. This chapter describes an implementation of a negotiation between two agents following a bilateral protocol and using one-step strategies from [Faratin and al. 1999]. The performance, efficiency and stability of the simulation in achieving a reasonable agreement are analysed. Protocols expressed in ANML can be further used to derive a set of possible paths to a goal state. An agent can then plan ahead using search strategies, path-finding, prediction, risk assessment, game theoretic and AI techniques, engendering partial, full planning or re-planning capabilities in an agent.

Chapter 9 summarises the achievements and conclusions from our work. This thesis shows that automated negotiation between rational and planning agents can be facilitated through our framework for representing and reasoning about protocols and their properties. Improvements and further research in this area are included.

Appendix A gives in more detail the semantics of multi-modal languages, the inference rules and axioms that apply to the normal system that is ANML. Appendix B provides a critical analysis of electronic commerce and object oriented frameworks. Findings from a case study are used to prioritise the requirements for e-commerce and describe the steps in a business transaction. The second part of appendix B presents a number of object based electronic commerce architectures with focus on OMG frameworks, including a proposed OMG negotiation facility.

1.4 Background and Related Research

This section references past and current research in agent negotiation in electronic commerce. The related work is divided under four headings: multi-agent systems (MASs), electronic commerce (EC), automated negotiation and decision making mechanisms. Automated negotiation occurs in the context of MASs and EC and is enabled through protocols and strategies.

1.4.1 Background: Multi-Agent Systems (MAS)

In practice, agents do not function in isolation from their environment which may contain other agents. In order to manage the complexities and dynamics of such systems, agents may be endowed with characteristics of coordination, [Gasser 1992; Malone 1986], cooperation, [Tennenholtz 2001], and social behaviour, [Sergot 1999]. "Multi-agent systems are social in character; there are properties of DAI systems which will not be derivable or representable solely on the basis of properties of their component agents", [Gasser 1998]. Social agency, [Conte and Castelfranchi 1995], involves abstractions from sociology and organisational theory to model societies of agents. Agents do not have complete and accurate information, which makes it difficult for groups to attain coherent global behaviour. However, agents can mitigate this by coordinating their actions and share knowledge through communication. Such *inter-agent communication* helps an agent to better achieve its goals or those of the society/system in which it exists – especially when resources are limited, [Huhns and Stephens 1999].

The limited horizons of agents in MASs require dynamic collaboration for distributed problem-solving, facilitated by an agent's local autonomy. This gives rise to dynamic

organisational structures such as virtual enterprises or teams. Organisations are heterogeneous, complex and dynamic systems that are comprised of multiple agents joining together to achieve, for example, higher performance, [Gasser 1998]. Mathematical and computational methods have been used to study human and agent organisations, [Galbraith 1973; Lendaris 1964; Manning 1977] and a large number of models have resulted, each focussing on specific aspects of an organisation [Cohen and al. 1972; Corkill 1983; Pentland 1995]. For example, teams may be defined as groups of agents that are restricted to having a common goal within the team. See [Grosz and Kraus 1996; Rich and Sidner 1997; Tambe 1997] for research in modeling teamwork. Groups of agents may also adopt joint intentions for cooperation, [Cohen and Levesque 1991], where involvement in a joint task requires informing others in the group of relevant commitments and/or de-commitments. Agents may as well achieve cooperation by learning and communicating about each other's abilities, [Huhns and Weiss 1998; Sen 1996 and 1998; Weiss 1998].

Formal modelling of multi-agent systems

Formal methods are useful for reasoning at a higher level than the implementation level and provide correctness of programs. Formalisation techniques are generally used in language specification, verification of programs, for reasoning about knowledge and action or as meta-languages to specify behavioural properties of agents, [Singh, Rao and Georgeff 1999]. Classical logic and other forms of modal logic may be used to represent information about an agent and its environment. For example, deontic logics, [Prakken and Sergot 1997], help when reasoning about agent's mental states such as knowledge, beliefs, desires, goals and intentions. AI research has become interested in reasoning about knowledge and beliefs as in epistemic logic and doxastic logic, [Meyer and Van der Hoek 1995; Fagin and al. 1995].

1.4.2 Agent Mediated Electronic Commerce and Markets

Statistics in June 2001, [Cox 2001], show that about 100.2 million people in the U.S. or nearly half of the adult population there with access to the web have made a purchase online at one time or another. Research is being performed on a range of associated issues in electronic commerce such as: brokering, [Decker and al. 1997], security, [Tahara and al. 2001], trust, [Marsh 1994; Robles and al. 2001], negotiation,

[Jennings and Wooldrigdge 1997; Rosenschein and Zlotkin 1998; Sandholm and Lesser 1995], ontologies [Farquhar and al. 1996; Huhns and Singh 1997], multimedia, law, [Bench-Capon 2001], and payment, [Wang and al. 1998].

Real world economics and market strategies can help in agent electronic commerce to analyse protocols and strategies. Economics concepts may be embedded in designing algorithms for economic transactions. [Wellman 1998] presents a market model for distributed design, where market mechanisms and social constructs are used to analyse complex computational systems. There have been various other efforts to exploit market economics for distributed electronic commerce frameworks, [Clearwater 1995; Sandholm and Suri 2001; Wellman 1995].

1.4.3 Automated Negotiation

Negotiation is a form of decision-making where two or more parties jointly search a space of possible solutions with the goal of reaching a consensus. Through negotiation, computational agents are able to find better deals in strategically complex environments possibly containing parallel transactions. A negotiation can be competitive or collaborative and can occur in a closed or open marketplace. In competitive negotiation, the parties are self-interested and have conflicts of interest. Collaborative negotiation involves agents co-operating to achieve their goals. In a closed marketplace, users enrol in the marketplace, agree to a certain set of rules and the set of users is predefined. In an open marketplace, agents can enter and exit at any time and do not need to agree to rules.

The contract-net protocol, [Smith 1980], embodies the first attempts at automated negotiation. Kasbah, [Chavez and Maes 1996], Auctionbot, [Wurman 1998], the FishMarket Project, [Rodriguez and al. 1998a], and Tete-a-Tete, [Chavez and Maes 1996], are trading agents that in addition to brokering perform some basic negotiation, but they are semi-autonomous and concentrate on price. More advanced forms of negotiation include multi-issue and flexible interactions. This section references research on the features and mechanisms for automated negotiation. Contracting and commitment are the last stages of a negotiation and are addressed in [Sandholm and al. 1999b; Kraus 1996].

For agents to interact meaningfully in negotiations, common languages, ontologies and protocols are required regardless of an agent's private strategy. Protocols are used to coordinate the activities of a group of agents to satisfy an agent's goal and the group's goal. A negotiation protocol is essentially a (possibly) pre-determined sequence of messages to form a conversation. It is a pattern of message exchanges that two or more agents agree to follow in communicating with one another. The need for protocols for interaction in a group is increasingly recognised, [Labrou and Finin 1997b]. Interaction protocols have to be formally specified and implemented, yet relatively little work has been done so far on these two issues. This thesis answers the need for well-defined and sharable conversation protocols, with testable and desirable properties, that can be used to coordinate agents in accomplishing specific tasks. [Sandholm 1996] and [Rosenschein and Zlotkin 1998] survey protocols and discusses the constraints and game theoretic properties of protocols for multi-agent systems.

There are various types of negotiation protocols: auction, bilateral, multilateral, contract and argumentation amongst others. Combinatorial auctions, [Lehman and al. 1999], allow a bidder to submit bids for a combination of items. Several researchers have been trying to develop optimisation algorithms for combinatorial auctions, [Hunsberger and Grosz 2000; Rothkopf and al. 1998]. Negotiation by argumentation, [Kraus 2001a; Parsons and al. 1998], as another type of interaction, involves exchanging proposals that are backed by arguments that summarise the reasons why the proposal should be accepted. Threats and promises are the most common forms of arguments used in a human negotiation, [Boster and Mongeau 1984]. In its electronic counterpart, an argument aims to alter the mental state of a receiving agent and usually serves to modify the persuadee's set of intentions. Most of the argumentation frameworks are based on logical models of the mental states of the agents representing their beliefs, desires, intentions and goals.

Another requirement for agent interaction is an agent communication language, which is often a collection of messages consisting of speech-acts and the semantics of that language. There have been efforts for standardising agent languages for communication, [Labrou 2001; Poslad and Charlton 2001]. However formalising the semantics of communications has proved to be a longstanding challenge because more

than one view is possible. An ontology provides a shared concept of the world that can serve as the basis for communication and helps to build an understanding of the content in a message. Examples of multi-agent architectures including an ontology can be found in [Dowell and al. 1997; Farquar and al. 1996; FIPA 1997].

1.4.4 Strategies and Planning

While ontologies, protocols and languages are commonly known in a group of agents, an agent can choose its strategy privately for evaluating, generating and deciding on its next best course of action. It is similar to a decision process which an agent uses to determine its positions, concessions and criteria for agreement. A self-interested agent will choose the best strategy for itself. One step decision-making algorithms may be reused or extrapolated towards plan formation. Search algorithms help to solve path-finding problems, [Pearl 1984], constraint satisfaction problems, [Tsang 1993] and in game trees, [Shannon 1950]. [Yokoo and Ishida 1999] surveys such algorithms for problem solving and planning by agents. Distributed planning concerns distributing the process of planning among agents, each of which contributes pieces to the plan, until an overarching plan is created. [Durfee 1999] studies distributed planning, post- and pre-plan coordination and plan merging.

Game theory is useful in devising strategies and coalition formation, where which coalitions to be formed depend on maximising the overall utility of a group, [O' Shehory and Kraus 1999; Sandholm and Lesser 1997]. [Rosenschein and Zlotkin 1998] adopt a game-theoretic approach to analyse a negotiation in different domains and decide on a strategy on the basis of utility assignments to goals. They study equilibrium theory for maintaining stability in games, which is important for agreements and commitments. In comparison, Sycara's model, [Sycara 1990], uses case-based reasoning and optimisation of multi-attribute utilities. [Faratin and al. 1998; Rodriguez and al. 1998a] describe one-step decision making algorithms based on utility functions, trade-offs, imitative tactics and constraints on resources. [Sandholm and Lesser 1995] study automated negotiations with bounded rationality, in the context of the contract-net framework. However traditional game theory models are not suitable for competitive scenarios, because they assume full rationality and complete knowledge about the goals and strategies of one's opponents.

1.5 Statement of Contribution

A meta-language, ANML, is provided as a methodology for specifying and reasoning about negotiation protocols and their properties. It allows verifiable protocols to be expressed as intuitive and concise logical theories so that they can be understood and shared by goal-seeking, rational negotiating agents. Thus agents will be able to advertise the protocols they are willing to engage in or to adopt from other agents.

An analysis of existing agent systems, electronic commerce architectures and automated negotiation has led us to recognise the importance of common languages, ontologies and protocols for automated negotiation. Having identified the need for tools for formalising, sharing and extending protocols for interactions, we define (syntactically and semantically) a framework consisting of a meta-language, ANML, for supporting automated negotiation through AI techniques. ANML, as an extension of propositional dynamic logic, is a meta-language for specifying abstract theories of negotiation and for validating conversation protocols. Expressing a protocol as a logical theory in ANML enables an agent to find equivalence between different protocols and enable protocol extension and composition by agents. The protocols represented can also be made to exhibit safety and liveness properties in addition to correctness. Safety and liveness axioms are specified in ANML and demonstrated using a bilateral protocol as example.

In addition, this thesis gives the conditions for attaining consistent joint knowledge and beliefs, even in a fallible communication medium. in a group while each agent has individual mental states. Last but not least, the logical essence of our framework allows reasoning about the processes and states of a negotiation towards goal states by agents equipped with planning capacities to facilitate the combination of planning with other DAI research on strategic decision making.

Our approach is compared with methodologies such as statecharts and AUML and shows errors in protocols expressed in these notations. Existing protocols, as case studies, are verified and their corrected versions are specified in ANML. It is shown that the meta-language ANML has enough expressiveness to specify dynamic multilateral *m-n* negotiations.

2 Agents for Negotiation

2.1 Introduction

Object-oriented frameworks are not particularly suitable for supporting strategic decision making in unpredictable environments as in automated negotiations. (See Appendix B and [Cunningham, Paurobally and al. 1998] for a critical analysis of electronic commerce and object oriented frameworks). New software architectures based on autonomous and intelligent software agents promise more elegant and evolutionary techniques for supporting and enabling electronic commerce than object architectures. The software agent paradigm grew from AI and distributed problem solving systems and there are a variety of proposed definitions of what an agent is, without a commonly agreed one. Essentially, an agent is situated in an environment, senses its environment and acts autonomously and proactively upon it, continuously over some period of time [Jennings and Woodridge 1997]. They respond to changes in environment, interact with other agents and take initiative where appropriate. Agents can reason about stored knowledge to act without the intervention of a user in a dynamic environment.

This chapter describes the characteristics of agents including non-logic, logic-based and hybrid agents. Hybrid agents are built out of two or more types of agents. This chapter surveys various agent design concepts, agent architectures and applications, more specifically those in electronic commerce and automated negotiation.

2.2 Characteristics of Agents

[Wooldridge and Jennings 1995b] define an agent as being both autonomous and flexible in some environment. Agent sense and act over a possibly dynamic

environment using their own initiative. Examples of environments in which agents may be situated include the physical world, a user, a society of agents or the Internet. In addition to autonomy, agents need to respond in time to changes, that is be *responsive*. The four basic attributes of an agent according to [Wooldridge and Jennings 1995b] are *autonomy*, *responsiveness*, *proactiveness* and *social awareness*. *Proactiveness* means anticipating events and taking initiative where appropriate. *Social* agents are able to interact with other agents via an agent communication language to complete their goals or to help others with their activities. The autonomy and proactivenss capabilities of agents render them more suitable for problem solving and strategic decision making in complex business domains.

Other optional characteristics of an agent include *learning*, *sincerity*, *mobility*, *opportunistic*, *deliberation*, *reactivity*, *intentionality*, *rationality*, *benevolence*, *truthfulness* or *co-operation*. Learning involves an agent being adaptive and changing its behaviour based on its previous experience or observations. Sincerity, truthfulness and benevolence behaviours entail an agent does not lie and is not malicious. Mobile agents are capable of moving around networks. Opportunistic behaviour in an agent shows seizing environmental opportunities for satisfying its goal. A deliberative agent has an explicitly represented, symbolic model of the world, and decisions are made via logical reasoning and symbolic manipulation. However deliberative agents find it unfeasible and slow to achieve complete knowledge of the world. Reactive agents have no symbolic reasoning and do not plan ahead. Intentional agents have beliefs, desires and intentions and act according to their intentions. A rational agent acts in a goal-directed way to satisfy its goals according to its internal states. Co-operative agents collaborate with other agents for satisfying the group's goals.

2.3 Non-logic Based Agents: Reactive Agents

A reactive agent does not have a central symbolic world model and does not use symbolic reasoning. Such agents do not plan ahead and their actions depend on what happens at that time. Reactive agents tend to operate on representations which are close to raw sensor data. [Brooks 1990] supports that representations should be grounded in the physical world so that the world is its own best model obviating the need for symbolic representations. A typical architecture consists of parallel and

independent modules that are responsible for specific tasks, and which all interface to the world rather than interface to each other. Communication between the modules is minimised and of a low-level nature. Reactive agents act/respond in a stimulus-response manner to the present state of the environment in which they are embedded, [Brook 1991]. Complex patterns of behaviour and a global behaviour are supposed to emerge from interactions with the environment when the ensemble of agents is viewed globally. The complexity of an agent society and its behaviour is a consequence of their interaction with the environment, not due to the complexity of the agents themselves.

2.3.1 Brook's Subsumption Architecture

Brooks proposes the subsumption architecture for reactive agents, [Brook 1991]. It is composed of a hierarchy of parallel task-accomplishing layers where each layer competes with the other layers to exercise control over the robot. Lower layers show primitive kinds of behaviour, such as avoiding obstacles and have precedence over higher layers.

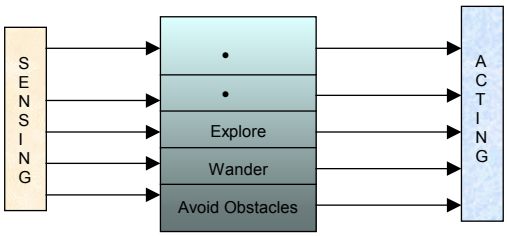


Figure 2.1 Brook's Subsumption Architecture

Each layer shows a particular behaviour and pursues a purpose by interacting with the world. The layers must decide when to act for themselves. This approach gives an incremental behaviour from simple systems to complex systems. A small piece is built and interfaced to an existing working complete system. The advantages of this architecture lie in the speed of reactions and robustness. There are no complex representations to maintain and reason about, reducing latency. Some layers may still be functioning while others are incapacitated.

However to explore the world blindly is not an intelligent search and can be costly and time consuming. A solution is to use partial models of the world where individual layers extract only aspects of the world relevant to their task. Other open

issues concern the scalability of the system while keeping complexity reasonable and how complex are partial representations of the model. Higher layers can be endowed with learning capabilities. There are questions about how the different layers rely and interface/interfere with each other, the interaction between multiple robots and the emergence of complex behaviours.

2.4 Hybrid Agents

Hybrid agents are built out of two or more types of agent architectures, for example deliberative and reactive. The benefits from combining architectures are possibly greater than that obtained from a single type of agent. Often, the reactive component is given some precedence over the deliberative one for action, so as to provide a rapid response to important environmental events. Hybrid agent architectures are usually layered e.g. TouringMachines, [Ferguson 1992], and InteRRap, [Müller 1996]. Higher layers deal with information at increasing levels of abstraction. For example, the lowest layer might output actions directly on obtaining raw sensor data, while the uppermost layer deals with long-term goals. The three asynchronous layers in the InteRRap deliberative-reactive architecture are divided in three categories: *behaviour-based*, *local planning* and *co-operative planning*. The reactive part, consisting of situation-action rules, is implemented in the behaviour-based component and supports efficiency, reactivity and robustness. This layer has to provide fast situation recognition to react to time-critical situations. The other two components for planning allow for more deliberation.

The drawback with hybrid architectures concerns combining different types of interacting subsystems cleanly, in a well-motivated control framework. Existing applications remain domain specific.

2.5 Logics Based Agents

Logics can help to conceptualise and develop agent systems: it helps in formalising the knowledge, beliefs and goals of an agent, the perception of its environment, its reasoning component, the construction of plans, the generation of timely and appropriate reactions and the process of negotiation and communication amongst agents. Formal methods are useful for reasoning at a higher level than the

implementation level and providing correctness of design. Such methods may be used in specifying a language for reasoning about knowledge and action or for specifying behavioural properties through meta-languages.

One of the main contributions in this area is the BDI architecture, [Rao and Georgeff 1995], which consists of an interpreter and deliberating and planning modules. Another type of logic-based architecture contains deliberative agents. A deliberative agent has a symbolic model of the world and develops plans and decisions via logical reasoning, pattern matching and symbolic manipulation. However deliberative agents encounter the transduction problem, which is how to translate the real world with complex processes into an accurate, adequate symbolic description, in time for that description to be useful, [Wooldridge and Jennings 1995a].

An agent in a group has to consider both its knowledge and beliefs and, if it is a social agent, what others may know or believe. Such reasoning can become complicated, but common knowledge is important in a group in order to achieve a consensus, [Fagin and al 1995]. Reasoning about knowledge and common knowledge may be performed through epistemic logic, [Meyer and Van der Hoek 1995] and [Fagin and al 1995], where an agent's beliefs can be characterised as a set of possible worlds and *epistemic alternatives* describe the possible worlds given its beliefs. A proposition holding in all the epistemic alternatives of an agent's world is believed by it. Logic-based agent systems may alternatively use event calculus, [Kowalski and Sergot 1986], inductive logic programming, [Muggleton 1999] and abduction, [Kakas and al. 1992], for reasoning. Below we describe three well-known logic-based architectures.

2.5.1 Shoham's Agent-Oriented Programming

In Shoham's agent-oriented programming (AOP) framework, [Shoham 1993], an agent's state consists of mental components such as *beliefs*, *capabilities*, *choices* and *commitments*. Shoham's AOP system consists of three primary components:

- A formal language for describing mental states through modalities.
- An interpreted programming language to define and program agents with primitive communication commands.

- An *agentifier* to convert neutral devices into programmable agents.

Shoham [1991] defines a language for beliefs, obligations and capabilities, called *Agent0*. Several properties about the modalities are given e.g. beliefs and obligations are internally consistent, agents are aware of their obligations and the persistence of mental states. *Agent0* has two basic mental categories, beliefs and commitments. The actions of an agent are determined by its decisions or choices. An agent's decisions depend on its beliefs referring to the state of the world, on the mental state of other agents, on prior decisions, on its capabilities and other agents' capabilities.

Each agent is controlled by its private program. An agent's program consists of a definition of its capabilities and initial beliefs, fixing the time grain and a sequence of commitment rules. An agent's program also contains primitive operations and IO instructions for communicating with other agents. The initialisation part of an Agent0 program defines the capabilities of that agent, its initial beliefs and commitments. Commitment rules determine how commitments are added over time according to the current mental state and incoming messages. Actions to which an agent is committed always refer to a particular future point in time. An interpreter verifies whether a requested action is in the agent's capabilities and checks that contradictory prior commitments are absent. The interpreter reads the current message and updates the mental state, then execute the commitments at the current time, possibly resulting in further belief change.

In *agent0*, agents are naïve and accept any information, retracting previous beliefs if necessary. An agent's beliefs database contains private and common beliefs. The result of informing is common belief between the informer and the agent. Beliefs change can remove capabilities. Restrictions such as naïve belief-revision must be relaxed and update of mental state should be able to last less than some fixed time grain. Agents are also unable to plan and communicate requests for action via high-level goals. The framework could be extended to deal with more mental categories, with agent societies and to support persistence of mental states. The notion of mental state has to be enriched by adding intentions.

2.5.2 BDI agents

As stated earlier, Rao and Georgeff [1991] developed a logical framework for an agent theory based on three primitive modalities: beliefs (B), desires (D) and intentions (I). These three attitudes are represented with three dynamic data structures in their architecture.

The beliefs database contains an agent's knowledge about the state of the world, its own internal state, other agents and their mental states. An agent's desires correspond to the tasks allocated to it. Goals are conditions for an agent to achieve, test, maintain and wait for. An agent will not in general be able to achieve all its desires and must therefore decide on some subset of its desires and commit resources to achieving them. An agent's intentions represent the chosen desires that it has committed to achieving. There are compatibility functions, such as belief-compatible, which are critical in enforcing formalised constraints upon an agent's mental attitudes. The abstract data structures for beliefs, desires and intentions can be updated and queried.

Another component in the BDI architecture is a plan library. This is a set of plans specifying courses of actions that an agent may follow to achieve its intentions. Each plan consists of an *invocation condition*, which specifies upon which events the plan should be triggered, a *context condition* which specifies under what situations the plan applies and a *body* which describes the steps of the plan, [Rao and Georgeff 1995].

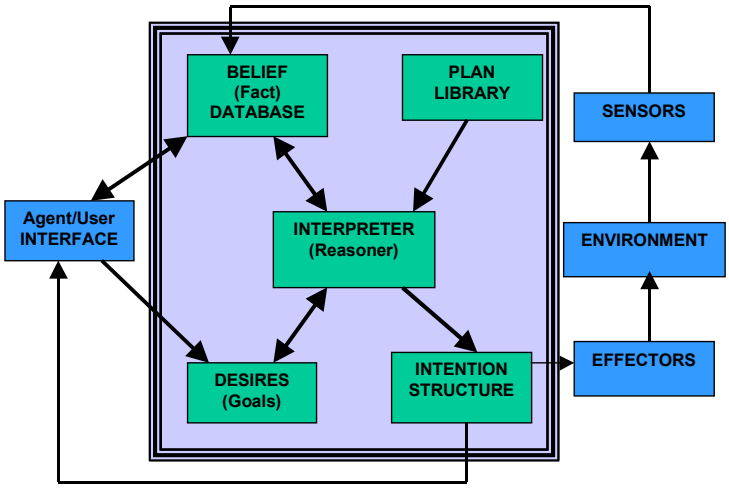


Figure 2.2 PRS BDI Agent Architecture

In Figure 2.2, the interpreter performs belief updates from observations made of the world and generates new desires. Changes in the system's goals and beliefs invoke a set of possible plans, some of which are chosen and placed in the intention structure. The interpreter then selects an action to perform on the basis of its agent's current intentions and knowledge. This can result in establishment of a new sub-goal or belief. These newly established goals and beliefs trigger new plans to be selected and placed in the intention structure.

An option generator in the interpreter deduces the possible actions of an agent. Reducing the options generated by the option generator yields various notions of commitment and results in different behaviours of an agent. A *blindly-committed* agent is one that maintains its intentions until it believes that it has achieved them. A blindly-committed agent will eventually come to believe it has achieved its intentions no matter with what information it is supplied or it discovers. *Single-minded* commitment allows an agent to maintain its intentions as long as it believes that they are still options. A single-minded agent will not drop its intentions as long as it believes its intentions are still achievable. A single-minded agent is committed to its goals but is open to changes in its beliefs. An *open-minded* agent is one that is prepared to change its goals as well as its beliefs but otherwise maintains its intentions as long as these intentions are still part of its goals, [Rao and Georgeff 1995].

2.5.3 Computational Tree Logic (BDI CTL*)

The logics of CTL and CTL*, (Computational Tree Logic) [Clarke and Emerson 1981], extend the possible worlds model, [Kripke 1963], to address temporal aspects. CTL comprises of state and path formulae but is not expressive enough for expressing the relation between a state formula and a path formula. CTL*, [Emerson 1990], extends CTL by making a state formula an admissible path formula.

In CTL*, the world is modeled as a time tree with a branching future and a single past. A particular time point in a particular world is called a situation. Events transform one time point to another. Compound events map to non-adjacent time points, so allowing the modeling of partial plans.

BDI_{CTL*} theory is an extension of CTL* by combining branching time logic with conventional modal operators for beliefs, desires and intentions, applied to rational agents. BDI_{CTL*} is two dimensional; a dimension of possible worlds with *Bel*, *Des* and *Intend* modalities and a dimension of time along which the temporal formulae are evaluated. A *situation* is an ordered pair consisting of a world and a state. Every situation is associated with a set of belief-accessible worlds that are time trees which an agent believes to be possible. An agent believes the actual world to be one of its belief-accessible worlds. Multiple belief-accessible worlds result from an agent's lack of knowledge about the state of the world. The branching future represents the choice of actions available to an agent.

For each belief-accessible world, *w*, at a given moment, *t*, there must be a goal-accessible world that is a sub-world of, *w*, at time, *t*. A goal-accessible world represents a possible world if the agent's goals were satisfied. Desires can be inconsistent with one another while goals are specified to be consistent. Goals are chosen desires of an agent that are consistent and are believed to be achievable.

Intentions are represented by sets of intention-accessible worlds that the agent has committed to attempt to realise. For each goal-accessible world, *w*, there must be an intention-accessible world that is a sub-world of *w* at time *t*. An agent can only intend some course of action if it is one of its goals. Intention-accessible worlds must be compatible with goal-accessible worlds.

Two problems in the possible worlds semantics of CTL* are that of knowing all valid formulae and of knowledge/belief being closed under logical consequence, known as the logical omniscience problem, [Reichgelt 1989]. Levesque [1984] proposes making a distinction between explicit and implicit beliefs as a solution to the logical omniscience problem, where an agent has a relatively small set of explicit beliefs, and a very much larger (infinite) set of implicit beliefs, which includes the logical consequences of the explicit beliefs. However his solution does not allow for nested beliefs and quantification. The *side effect* problem in the goal-accessible world approach predicts an agent has a goal out of the logical consequences of its goals. A popular example is that an agent might have a goal of going to the dentist, so the necessary consequence is suffering pain but it does not have as goal to suffer pain.

2.6 Agent Design Concepts

2.6.1 Mobile Agents

Mobile agents are processes that are capable of moving around networks such as the WWW, interact with foreign hosts to gather information, transact on behalf of their owners, access remote resources or cooperate with other agents and then return to their owners. They can encapsulate scripts and be dispatched on demand. In migration, a mobile agent moves between nodes and uses collected information to adjust its behaviour and progressively accomplish its task. Mobile agents reduce communication costs, which would otherwise have been time-consuming and would have caused traffic on the networks. They are useful when the local resources such as processing power and storage on the local machine are limited. They allow asynchronous computing and support electronic commerce with their distributed architecture.

Telescript applications, [White 1994], consist of mobile agents operating within a world or cyberspace of places. Telescript however requires the programmer to learn and work with a complex object-oriented language and a complex security model. Significant issues to be addressed, when agents are mobile, remain authentication, privacy and security of the hosts and agents. It is not obvious that mobile agents consume less bandwidth since it may involve transferring complex code. Either the mobile agent is complex and heavy, requiring more bandwidth, or it has limited capabilities.

2.6.2 Learning agents

Learning technology initially consisted of standalone architectures and then evolved to neural nets and machine learning. The area of learning in multi-agent systems has received increased interest, [Huhns and Weiss 1998; Sen1996; Weiss 1998]. Agents may learn to cooperate by learning and communicating about each other's abilities. Learning can be viewed as a method to reduce communication, although communication may be used to continue or refine learning, [Sen and Weiss 1999].

There exist different approaches to learning for the purpose of coordination: reinforcement learning, [Kaelbling and al. 1996], Q-learning, [Watkins 1989], isolated learners, [Sen and al. 1994], interactive learning, [Weiss 1993], or machine learning, [Mitchell 1997].

[Mitchell 1997] defines machine learning as the study of computer programs that improve through experience their performance at a certain class of tasks, as measured by a given performance metric. A machine learning system aims at determining a description of a given concept from a set of concept examples provided by the teacher and from the background knowledge. Learning from an incomplete set of examples is called inductive learning. Inductive learning is based on an *inductive learning hypothesis* where any hypothesis found to approximate the concept well over a sufficiently large set of training examples will also approximate the concept over other unobserved examples, [Mitchell 1997]. A learning algorithm may use positive and negative examples of a concept and examples of several concepts, depending on the aim of learning. In supervised learning, examples are annotated with their corresponding concept, while in unsupervised learning the data is provided along with the definition of a language which has to be used to represent it.

In reinforcement learning, an agent adapts its decision process based on environmental feedback from its choice of actions. Reinforcement learning includes techniques such as Q-learning and relational reinforcement learning, [Kaelbling 1996]. In Q-learning, an agent's decision procedure is specified by a policy π that maps states into actions. A reward function deals with environmental feedback to maps states into numerical rewards. The goal of Q-learning is to compute an optimal policy π^* that maximises the reward that an agent receives. Storing all Q values can lead to memory space problems and a neural net may be used as a function approximation. However this can lead to more complex update mechanisms. In some domains, it may be difficult to specify a numerical reward function. Relational reinforcement learning combines reinforcement learning with relational learning or inductive logic programming.

2.6.3 Interface and Information Agents

An *interface agent*, [Lieberman 1997], is a program that employs reasoning to provide assistance to a user dealing with a particular application e.g. an operating system or an auction applet. It acts as an autonomous personal assistant collaborating with the user in accomplishing some task in the same environment. An interface agent observes and monitors actions taken by a user via an interface, adapts over time to a user's preferences and habits and suggests improvements in performing the task. It learns through observing and imitating its user, obtaining feedback, instruction forms and asking other agents for advice. However, cooperation with other agents is usually limited to asking for advice without negotiating. Learning modes are typically by rote or parametric though there can be evolutionary learning.

An *information agent* accesses one or more distributed information sources for information search and retrieval. They manage, collate and manipulate obtained information in order to answer user queries. Information agents manage the current growth of information on the Internet, helping in information management and avoiding information overload.

2.7 Multi-Agent Systems

Multi-agent systems, (MAS), contain two or more agents which may or may not collaborate. MAS address problems that are too complex for a centralised single agent with limited resources. They can analyse distributed information sources and expertise for distributed problem solving, e.g. air-traffic control or distributed sensor networks. MAS encourage modularity, reusability, flexibility, reliability and increased speed through parallelism. However issues such as interconnecting, interoperation, co-operation of multiple systems including legacy systems and understanding interaction among human societies still remain to be resolved.

A heterogeneous agent system contains at least two or more different types of agents, including hybrid agents. The agents may handle their own co-ordination or groups of agents may rely on a control program to achieve co-ordination. In the federated approach, agents communicate only through intermediaries called *facilitators* or *mediators*. It resembles brokering whereby facilitators locate other agents on the

network capable of providing various services. Facilitators establish connection across networks and ensure correct interaction amongst agents.

2.7.1 ARCHON Architecture

The ARCHON (ARchitecture for Cooperative Heterogeneous ON-line systems) architecture, [Cockburn and Jennings 1995], supports decentralised and co-operating communities of problem-solving, loosely coupled, semi-autonomous agents. Partial sub-systems can be integrated into a larger coherent community.

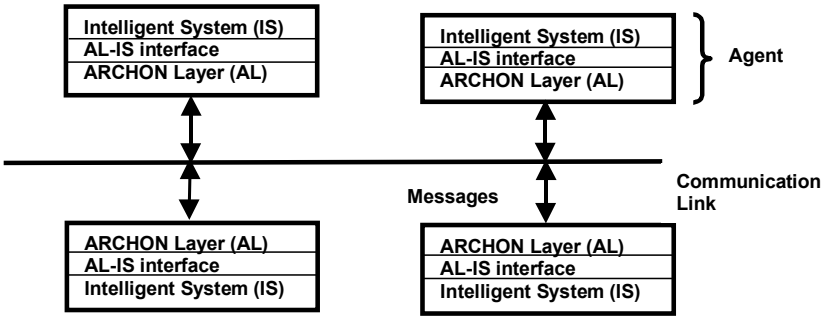


Figure 2.3 Structure of an ARCHON Community

An agent in ARCHON is able to control its own problem solving and to interact with other community members for co-orporation to solve the overall problem. Each agent consists of a communication module called an ARCHON layer, (AL), a reasoning module named an Intelligent System (IS) and an interface between the two components. Each agent controls its own IS and mediates its own interactions with other agents, controlled by the AL. An ARCHON layer allows agents to interact with each other through a standardised agent interface e.g. asking for information and requesting services. The system's overall objective is expressed in the separate local goals of each community member.

An AL invokes functions on the IS. An IS layer incorporates the planning and decision-making capabilities of the agent. The ISs can be heterogeneous - in terms of programming language, individual control algorithm, problem solving paradigms and platform. The differences are masked by a standard AL-IS interface. An ARCHON layer is made up of four modules, each of which is responsible for one aspect of the functional architecture. A *monitor* module is responsible for controlling the local IS. A *planning and coordination* module performs reasoning about the agent's role in the

wider co-operating community, deciding which actions should be taken in order to exploit interactions with others whilst ensuring that the agent contributes to the community's overall well being. An *agent information management* model is a distributed object management system providing information management services to co-operating agents. A *communication* module allows agents to communicate with one another using services based on the TCP/IP protocol.

2.8 Agent Mediated Electronic Commerce

Agents can help to overcome the barriers that stop consumers using the Internet for sophisticated means by making the Internet more reliable and easier to use. Personal user agents can customise an application for an individual user, making it more effective and productive, and therefore a better trading medium. Within the bounds of security requirements, an information agent can retrieve, analyse and make use of information available on internal networks, commercial data sources and WANs. Agents can be used in data mining, data analysis and intelligent decision-support tools. In the longer term of 4G telecommunication networks, agents may be delivered through new terminal devices and embedded in intelligent equipment. Agent capabilities for strategic reasoning can help in business to business trading for a long-term benefit between intelligent, communicative software agents, the information market and electronic commerce.

Industrial research is carried out on customisation of information content and enhancement of interaction, services and data analysis through agents. A customer in an agent-oriented market may be divided into three groups: individual, corporate and developer. Individual users employ personal agents for information retrieval and customised value-added services. Corporate users use co-operative agent technology in information retrieval, data mining, problem solving and decision-support tools. Developers use agent architectures to support agents across distributed systems. There are a number of factors that need to be considered for agent mediated electronic commerce:

- The performance problems that agents aim to reduce may worsen instead and prevent the use of agent-based services. User agents must be easy-to-use.

- Electronic commerce agents need to reassure potential customers that they are secure, trustworthy and respect the privacy of the personal information they hold. The W3 Consortium, the Electronic Frontier Foundation and CommerceNet have set up the E-Trust to develop a branding system that will enable Web applications to advertise that they comply with the consortium's standards for respecting the privacy of an individual's transactions and interactions.
- In both the individual and corporate sectors, the complexity of distributed agent systems brings its own problems. Such systems may be difficult to design, develop, deliver and maintain.
- Electronic commerce, and the agents that support it, are prone to several high-level risks such as technological failures and commercial failures - leading to a collapse of the market, failure of agent technology and misuse of agents.

2.8.1 The ADEPT Agent Architecture

Multi-agent systems can support business processes because the domain involves inherent distribution of data, the autonomy of components, sophisticated interactions and uncertainty in the solution. Since agents are autonomous there are no control dependencies between them, therefore if an agent requires a service from another agent, they must come to a mutually acceptable agreement about the terms and conditions under which the desired service will be performed. In [Jennings and al. 1996], such contracts are called service level agreements (SLAs). The parties become involved in a joint decision making process by verbalising their demands and moving towards agreements through concessions or searching for new alternatives. Client and server agents negotiate about a service until they come to a mutually acceptable SLA. An agent must keep track of its context since new agreements require that an agent reschedules its resources or currently scheduled services may fail and require an agent to replan its execution strategy.

There are three components in the ADEPT negotiation model – a *protocol*, *service level agreements* and a *reasoning model*. The protocol is based on speech-act performatives. It covers a phase of finding out services, a provisioning phase of coming to an agreement and a management phase of invoking the agreement. Models guide the negotiation behaviour by supporting the process of generating initial offers, evaluating offers, and counter proposing if offers are unacceptable.

The ADEPT architecture, [Jennings and al. 1996], defines an *agent head*, which is responsible for managing the agent's activities and interaction and an *agency* for representing an agent's domain problem solving resources. The agency can contain atomic tasks or other agents, thus allowing a nested agent system. The coupling between an agent and its agency is tight whereas the coupling between and agent and its peer is loose. An agent head has communication, service execution, situation assessment and interaction management modules. A communication module routes messages between an agent and its agency and between peer agents. An interaction management module enables the provision of services through negotiation. A situation assessment module helps assess and monitor the agent's ability to meet agreements. A service execution module allows management of services throughout their execution. Acquaintance models store agreed SLAs with other agents and lists of peers which can provide services of interest. The self-model of an agent stores SLAs committed to, descriptions of the services the agent can provide, run time information and generic domain information.

2.8.2 Middleware Agents

A *preference* encapsulates the types of information a requester is interested in. A *capability* defines the types of requests performed by a provider. The *connection problem* involves finding other agents on the market, which might have relevant information, preferences or capabilities. This problem is addressed through middle agents that match suppliers' services to customers' requirements while allowing the participants in a transaction to preserve their anonymity. They may act as trusted third parties to ensure that a transaction is carried out as agreed. There are a number of situations arising between a middle agent, a requester and a provider depending on what information is advertised, [Decker and al. 1997].

A blackboard agent is a middle-agent that keeps track of requests posted by requesters (or customers). Providers then query the blackboard agent for events they are capable of handling. A broker is a middle-agent that protects the privacy of both a requester and provider. A broker routes requests and replies appropriately without the requester or the provider knowing about each other. In a pure brokered organisation, brokers are generally known to all the agents. A matchmaker or yellow-pages agent is a

middle-agent that stores capabilities advertised by providers. The requesters can then query, choose and contact any provider they wish directly, with the possibility of subscribing for repeated queries.

Broker organisations include centralised economic markets where the privacy of both requester and provider are protected. Dynamic entry and exit of agents can be handled and provide an easy way for load balancing. However such organisations suffer from the need of agents to have static knowledge of the brokers. Also this can lead to a communication bottleneck since all requests and replies need to go through the brokers, which form a single point of failure that cannot be resolved through local caching.

Different types of middle-agent can be combined to form hybrid organisations. In a hybrid and brokered organisation, a matchmaker is used to find an appropriate broker. Providers query a matchmaker to find a broker with whom to advertise. Matchmaker organisations can become decentralised markets with the proper caching mechanisms. They offer preference privacy to requesters, who keep total control over their own decisions. Any change in preferences or capabilities can be acted on instantaneously. With local caching, this type of organisation is robust and degrades rather gracefully. A hybrid system consisting of matchmakers and brokers allows capitalising on the lower overhead and efficient load balancing of a brokered system while retaining the dynamic capabilities and greater robustness of a matchmaker system. In case of broker failure, a hybrid system can continue to function by trading off privacy.

2.9 Automated Negotiation

Most online commercial sites belong to the first generation of e-commerce applications with client and server software; humans are still involved at both ends. In current online trading sites, the price and other aspects of a transaction are usually fixed, except in auctions. Negotiation is an important aspect of commerce. Two or more parties multilaterally bargain resources for mutual intended gain, using the tools and techniques of negotiation. In real life, we see examples of negotiation in auction houses and in bargaining. Online auctions such as Ebay and Amazon remove the constraint of all participants being co-located but still require users to manage their

negotiation. To reduce transaction costs and to automate time-consuming tasks, software agents that are personalised and continuously running may be used. Automated negotiation saves time and computational agents are able to find better deals in strategically complex environments.

The *contract-net protocol*, [Smith 1980], embodies the first attempts at automated negotiation through calls for proposal between contractors and manager nodes. Auctionbot, [Wurman and al 1998], the FishMarket framework, [Rodriguez and al 1998a], Kasbah, [Chavez and Maes 1996] and Tete-a-Tete, [Maes and al 1999], are advanced trading agents that in addition to product and merchant brokering perform some basic negotiation. Auctionbot and the FishMarket framework are concerned only with auctions and users have to encode their own bidding strategies. Kasbah is a web-based multi-agent electronic marketplace where users create buying and selling agents for transaction. A user wanting to buy or sell a good creates an agent, gives it some strategic direction, and sends it off into a centralised agent marketplace. Each agent's goal is to make the "best deal" possible, subject to a set of user-specified preferences such as a desired price, reservation prices, and a deadline. Negotiation between buying and selling semi-autonomous agents in Kasbah is bilateral. After buying agents and selling agents are matched, buying agents offer a bid to sellers.

All of the above applications focus on price. Negotiation may involve more that just interacting over price. Different users may have different preferences and many are interested in qualitative and quantitative aspects, other than price of a purchase. Agents should be able to negotiate over a variety of issues, find an agreement that satisfies all the parties and prepare contracts on behalf of their owners.

Inexpensive communication infrastructure and the rise of virtual enterprises favour automated negotiation. Standardised protocols and methods are needed for the automation or partial automation of negotiation as a step towards agreement resolution. Negotiation support systems are software programs for helping human make better decisions and negotiate more effectively. An agent has a set of tasks and resources. However, these systems require near-constant human input for both initial problem setup and final decisions. Although some agents may be taught strategies

through machine learning or genetic algorithms, humans are still required to delineate environment rules for optimal agent strategies to follow.

2.9.1 Requirements for Negotiations

An architecture supporting negotiation needs agents that clearly and rationally represent interests and information within each party's bargaining platform. Learning and training an agent to learn user preferences and strategies are useful to achieve its goals. A party may even create several agents with the same bargaining agenda to negotiate with several other parties simultaneously. Agent communication languages, ontologies and protocols enable agents to interact meaningfully in negotiations. For an agent to find the best deal, it needs negotiation protocols, strategies and plans. Contracting and commitment as the last stages of a negotiation are described in [Sandholm and Lesser 1995; Kraus 1996]. Sandholm and al., [Sandholm and al. 1999b] developed a framework where an agent can unilaterally decommit to a contract by paying a predetermined penalty.

An ontology is a representation of some part of the world. It provides a shared concept of the world that can serve as the basis for communication and helps to build an understanding of the content in a message. Ontologies are included in several multi-agent architectures, [FIPA 1997; Farquar and al 1996; Dowell and al. 1997]. The proper formulation of negotiation involves translating human preferences and issues into terms that can be meaningfully analysed, communicated and bargained. Buyer and seller preferences may be mapped into coherent utility functions. For that, a clear and unambiguous ontology is needed for example using KIF, [Finin and al 1993] and Ontolingua, [Farquar and al. 1996]. Both a seller and a buyer must regard a message and the item under negotiation as essentially the same. Conflicts within requests and preferences must be detected and resolved.

The problem of ethical issues has to be resolved for the adoption of agent applications in commercial transactions. An agent needs to maintain privacy when acting on the behalf of an owner who wishes to remain anonymous. In addition, agents should be reliable and discouraged to act badly towards other agents. An agents should follow some etiquette when in a society of agents such as identifying itself, moderating the pace and frequency of its requests to a server, sharing information with others,

respecting the server's authority, being accurate and truthful and not seeking to harm the server. Another problem is unpredictable behaviour of an agent from interactions. Shared protocols and ontologies may regulate the behaviour of an agent.

2.9.2 Agent Communication Languages

Agents need to communicate in a negotiation and this requires that they understand each other with a common language and protocol. They have to agree on how they send and receive messages, what the messages mean and how the conversation is structured between agents. In an agent communication language, the terms and the mechanisms used are at a high level of communication for interaction and co-operation. The contents of a communication are meaningful statements about the agents' environment, knowledge or actions. KQML, [Finin and al. 1993] and FIPA ACL, [FIPA 1997], characterise an agent as possessing mental attitudes such as *beliefs*, a set of propositions considered as true and *uncertainty*, a set of propositions accepted as not known. Agents have intentions, denoting a choice or set of properties of the world which the agent desires to hold and which are not currently believed to be true.

KQML (Knowledge Query and Manipulation Language), [Finin and al. 1993], is used to communicate attitudes about information such as *querying*, *stating*, *believing*, *requiring*, *achieving*, *subscribing* and *offering*. A KQML message is called a *performative* in that the message is intended to perform some action by virtue of being sent. Each agent appears on the outside as if it manages a knowledge base (KB). Communication with an agent is with regard to its KB e.g. questions and statements about what a KB contains, requests to add or delete statements from the KB, or requests to use knowledge in the KB to route messages to other agents. Performatives are grouped in discourse, conversation, networking and facilitation performatives. A message structure may optionally contain the message sender's name, a recipient's name, the content of the message (to express propositions, objects and actions), reply information, a content language and the ontology of the content. In each domain of KQML-compliant agents there is at least one agent with a special status called *facilitator* that can handle networking and facilitation performatives. Agents advertise to their facilitator, that is they send advertise messages to their facilitators to announce the messages that they are committed to accepting and processing. Network and

facilitation performatives allow agents to find other agents that can process their queries as facilitators can advertise messages community-wide. The KQML specification has been under much criticism in that many of its performatives are redundant, ambiguous and overlap in their functions. A formalisation of the KQML specification appeared in a later version, [Labrou and Finin 1997], to give the semantics of the performatives and to clarify their functions.

As a remedy to KQML, FIPA (Foundation for Intelligent Physical Agents), [FIPA 1997], has proposed two kinds of specification:

- *normative* specifications that guide the behaviour of an agent and support interoperability with other FIPA-specified systems
- *informative* specifications of applications for guidance to industry on the use of FIPA technologies.

The normative part contains specifications for agent management, agent communication language and agent/software integration. The FIPA Agent Communication Language (ACL) is based on speech act theory, [Searle 1969]. Messages are actions or communicative acts which can influence the actions of other agents. Every communicative act is described in both a narrative form and through formal semantics based on modal logic. The semantics are intended to give a deeper understanding of the meaning of the language and to be an unambiguous reference point. FIPA has also defined a number of protocols for patterns of conversations through an extension of UML called AUML, [Bauer and al 2001]. These range from simple requests for performing an action and querying, to setting up contracts and exchanging messages in auctions. Despite the semantics provided in the FIPA framework, communicative acts are still ambiguous and can lead to unexpected effects. Some researches, [Wooldridge 1998], deems communicative acts to be specific and internal to the agents in a group. There have been substantial efforts for standardising FIPA ACL for communication, [Poslad and Charlton 2001].

2.9.3 Negotiation Protocols

Negotiation protocols specify sequences of messages, possibly known to all the parties in a negotiation, that are to be followed in order to progress and to achieve a consensus. A negotiation protocol is a (possibly) pre-determined sequence of

messages to form a conversation. It is a pattern of message exchange that two or more agents agree to follow in communicating with one another. Protocols are used to coordinate the activities of a group of agents to satisfy an agent's goal and the group's goal. Protocols for cooperation, [Huhns and Stephens 1999], define ways to decompose and distribute tasks among agents. Although protocols need to be specified, verified, implemented and shared, relatively little formal work has been accomplished so far on these issues.

The contract net protocol is one of the earliest online protocols for agent interaction. It has been developed to specify problem-solving communication and control for nodes in a distributed problem solver, [Smith 1980]. The contract net protocol is a high level protocol for communication among the nodes to facilitate co-operative task-sharing. A collection of nodes is referred to as a contract net and a contract between two nodes enables the execution of a task. A low-level communication protocol allows reliable and efficient communication of bit streams between nodes. Task distribution is carried out by a negotiation process, stressing the utility of negotiation as an interaction mechanism. The contract net protocol addresses the connection problem where nodes with tasks to be executed must find appropriate idle nodes to execute those tasks, or vice versa. Solving the connection problem increases performance and affects resource allocation. A contractor may further partition a task and award contracts to other nodes. Control is distributed, as every node is capable of accepting and assigning tasks. A contract net is able to configure itself dynamically according to the positions of the nodes and the ease of establishing communication. However the contract-net protocol does not allow bidders to counter-propose better options, modify any of the service agreement parameters and a task manager has to devise a complete specification.

Rosenschein and Zlotkin [1998] apply game theoretic techniques to analyse the protocols governing the high-level behaviour of multi-agent systems. Protocols and strategies are analysed in three types of domain depending on the representation of an agent's goals. Their work shows that adjusting the protocols can influence an agent's private strategy and can encourage desired behaviours such as deception-free, efficiency and stability.

Negotiation by argumentation involves exchanging proposals that are backed by arguments that summarise the reasons why the proposal should be accepted. Threats and promises are the most common form of arguments used in human negotiations, [Boster and Mongeau 1984]. An argument aims to alter the mental state of a receiving agent and usually serves to modify the persuadee's set of intentions. Most of these frameworks are based on logical models of the mental states of the agents representing their beliefs, desires, intentions and goals, [Kraus 2001]. [Parsons and al. 1998] draws upon a logic of argumentation and implemented a form of dialectic negotiation.

2.9.4 Decision Making and Strategic Planning

In multi-agent systems, an interaction protocol may be public, but each agent chooses its own strategy. An agent's negotiation strategy chooses for the agent its next action. It is similar to a decision process which an agent uses to determine its positions, concessions and criteria for agreement. A self-interested agent will choose the best strategy for itself. Rosenschein and Zlotkin, [1998], examine a game-theoretic approach to analyse a situation and decide on a strategy by studying utility division in different domains. Sycara's model, [Sycara 1990], uses case-based reasoning and optimisation of multi-attribute utilities. [Faratin and al. 1998; Rodriguez and al. 1998a] describe various one-step algorithms for negotiation strategies, based on utility functions and constraints on resources. Also, coalition formation may be studied in an abstract setting like game theory, where which coalitions to be formed depend on maximising the overall utility of the group, [O' Shehory and Kraus 1999]. Some solutions however overlook communication costs and computation time. Tradeoffs have to be made between allocated computation, negotiation benefits and risk assessment. Traditional game theory models assume that all players are fully rational, have complete knowledge of details of the game and can reproduce equilibrium calculations of other agents. Evolutionary game theory, [Samuelson 1998], allows an agent to choose its strategies through a trial-and-error learning process.

Instead of one-step decision making, an agent may look ahead and use planning mechanisms for planning and predicting future paths to a goal state. Search algorithms address path-finding problems, [Pearl 1984], constraint satisfaction problems, [Tsang 1993] and game trees, [Shannon 1950]. These search algorithms

can be used for problem solving by agents and for constructing plans, [Yokoo and Ishida 1999]. Formulating a complex plan may require collaboration between a variety of cooperative planning agents. The process of planning can thus be distributed among agents, each of which contributes pieces to the plan, until an overarching plan is created, [Durfee 1999]. The latter explores strategies for task-sharing and result-sharing in heterogeneous systems. Distributed planning where either the plans or the planning can be centralised or distributed and post- and pre-plan coordination and plan merging are possible.

2.9.5 Auctions

Auctions represent a type of negotiation. With the increase in online auction sites such as Ebay, [Ebay 2001], a number of electronic transactions are conducted through auctions. Auctions often involve a set of pre-determined parameters and are therefore simpler to implement. Bargaining is restricted to price and the item for sale may be displayed with inspection facilities. Customers are usually capable of offering and counter-offering bids. A seller's strategy to assign awards is often made known. In an automated auction, an agent has to evaluate the trade-offs and implications of the variables. Well-known auctioning protocols include English auction or Dutch auction. [Sandholm 1996] surveys auction protocols and discusses the limitations of these protocols for multi-agent systems.

In English auctions, items are sold to the highest bidders and sometimes at the price of the second-highest bidder for equilibrium purposes. This prevents bidders from guessing each other's preferences and encourages them to bid their real valuation. A customer views current bids and can submit a bid. If his/her bid is surpassed, the customer can revise the bid and resubmit until the closing of the auction. Information is available to bidders and they can learn the valuation of other bidders. In a Dutch auction, an auctioneer starts with the highest price and lowers it until a buyer bids. In a first-price sealed bid auction, bidders submit a single irrevocable sealed bid. The bids are opened simultaneously and the winner is the highest bidder. Here less information is given to the bidder and he/she cannot dynamically update his/hers reservation price or base his bid on the bids of the others.

There can be hybrid auctions e.g. consisting of the English and the first-price sealed bid auctions. At first, such an auction is open, continuous and ascending as in an English auction. For the last few minutes when the bidder cannot be sure his/her bid will be received on time, the auction becomes a first-price sealed-bid auction. This kind of auction is advantageous to a seller since in the first phase, information is available and this raises the expected revenue. In the second phase, the optimal strategy is to bid one's own valuation when there are a large number of bidders.

Buyers and sellers have reservation prices, the maximum and minimum prices at which they will buy or sell respectively. A bidder wants to maximise the difference between the price paid and his/her valuation of the item. In the English auction, a bidder's optimal strategy is to bid up to his/her revised reservation price and then drop out of the bidding. In sealed bid auction, the strategy is to shade one's bid by some amount. A seller must optimise the number of items sold over a period of time, given holding costs and bidders distribution and strategies. As long as the holding cost is not too high, a seller would want to spread the items out as much as possible, selling fewer items at each auction and selling to the highest bidders possible. Information systems can keep track of bidding dynamics and detect profitable sessions. A seller can also kick-start an auction by placing phantom bids for prices above the highest current bid.

AuctionBot, [Wurman and al 1998], is an Internet auction server that allows anybody on the Internet to run an online auction. The users create new auctions to buy or sell products by choosing from a selection of auction types and specifying parameters (e.g., clearing times, method for resolving bidding ties, the number of sellers permitted, etc.). A seller specifies a reservation price after creating an auction and let AuctionBot manage and enforce buyer bidding according to the auction protocols and parameters. This partly resembles auctions at Ebay and Amazon.

Combinatorial auctions, [Lehman and al 1999], are cases when a bidder may submit bids for a combination of items. See [Hunsberger and Grosz 2000] and [Rothkopf 1998] for optimisation algorithms for combinatorial auctions.

2.10 Summary

Currently there are only basic forms of commercial automated negotiation that concentrate on price. Richer and more flexible negotiation applications, more scalable techniques for sharing information between heterogeneous agents, more elaborate resource management within agents and more flexible service scheduling algorithms are needed.

Shared languages, protocols and ontologies facilitate heterogeneous applications to attain a common understanding among themselves. A common protocol ensures that all participants following it will coordinate meaningfully and can expect certain responses from others. We acknowledge the importance of protocols in addition to an ACL in a negotiation since a protocol allows rational behaviour for deriving paths of actions towards a goal. Although conversations have become part of many multi-agent architectures, formalised conversation specification and implementation are needed for verification, goal-oriented reasoning and analysis of properties of a protocol and hence a negotiation. The semantics of a specification methodology have to meet these requirements. It is desirable to use formalisms with more support for concurrency and verification for sophisticated interactions. Our research aims to fulfil this need for tools for conversation specification, verification and sharing by proposing a meta-language for expressing protocols and abstract theories of a negotiation.

3 ANML – Agent Negotiation Meta-Language

3.1 Introduction

ANML (Agent Negotiation Meta-Language), is a meta-language for representing and reasoning about the states and processes of negotiation. ANML can be used to construct and validate properties of interaction protocols. Existing communication languages such as KQML and FIPA ACL focus on performative or communicative acts and their semantics. We recognise that in addition to standardised agent languages, shared protocols and conversations facilitate the interoperation of heterogeneous agents so that they can attain a common understanding among themselves. [Pitt and Mamdani 1999] argue that compared to the intentional semantics in FIPA ACL, use of common protocols reduces complexity, improves interoperability between components, supports reuse of recurrent protocols and facilitates verification of compliance with a standard. This thesis considers agent communication and negotiation through interaction protocols expressed in ANML. This chapter specifies the syntax of ANML, its semantics using Kripke structures, [Chellas 1980] and gives production rules and axiom schemata applicable to negotiation. First the need and contribution of ANML in specifying protocols are discussed, followed by a critical analysis of various formalisms that could have been used in specifying ANML. Finally ANML is specified as an extension of propositional dynamic logic, accompanied by an axiomatisation.

3.2 Motivation for ANML

There are a number of proposed frameworks for designing communication protocols. Many of them use finite state machines to do so, [Winograd and Flores 1986; Barbuceanu and Fox 1995; Kuwabara et al 1995; Wagner et al 1999; Elio and Haddadi 1999]. [Nodine and Unruh 1997; Pitt and Mamdani 1999; Martin et al. 1999; Lin et al.1999] use deterministic finite state automata to specify protocols. Other approaches for developing protocols include state charts, [Moore 1999] and Dooley Graphs, [Parunak1996]. Koning and Huget [2001] adopt a component-based approach and define a textual language to consider a protocol as an aggregation of micro-protocols. They show that Petri nets, [Cost et al. 1999] are not suitable for nesting and reuse of protocols. Noriega and Sierra [1996] use an extension of concurrent dynamic logic (CDDL) to specify multi-agent systems. An agent has a layered architecture and is represented in CDDL as a collection of formal theories in potentially different languages. In their multi-agent system, exchanges between agents are performed according to a dialogical framework. The latter determines the set of illocutions and predicates that can be used between the agents. Bridge rules model the translation between different communication languages and ontologies of agents. As an example of their framework, the Spanish Fish market auction is specified in CDDL. Chapter 4 specifies the protocol for a Spanish Fish market auction in ANML. FIPA AUML, [Odell and al. 2000], uses UML sequence diagrams to represent protocols.

Existing methodologies for specifying protocols are mostly diagrammatic or semi-formal. Most of them adopt finite state machines and automata as methodologies for reasoning about actions. However state transition diagrams lack expressiveness in representing constraints for transitions typed with an agent and are prone to ambiguity. Chapters 4 and 5 show the limitations of such informal notations as statecharts and AUML, leading to incorrect, incomplete and ambiguous protocols. Thus there remains a need for formal specification tools for protocols to facilitate the use and sharing of conversations between agents. A formal method supports verification and validation and addresses concurrency in interactions. This enables agents to define, in a shared methodology, the protocols they are willing to engage in, allow them to learn about other agents' protocols and to determine equivalency

between protocols. Existing protocols may be extended into new and more detailed versions or various protocols may be combined.

3.3 Possible Logics for ANML

It is difficult to find notation where processes and states are given equal status, let alone form the basis for a simple and rational calculus for an executable system. Traditional imperative programming languages, such a C and Pascal leave reasoning about programs to external axiomizations which impose complexity and incompleteness. Object oriented languages provide otherwise missing capacity for data abstraction, but without solving the problems of the need for a rational calculus. Process calculi like CCS [Milner 1989] do not consider state, while specification languages like Z, [Spivey 1988], are removed from executable systems. Hennessy-Milner Logic is impractical as it includes only primitive programs. It does not allow sequencing programs, iteration, recursion and if-then-else guards. In logic programming, we see demonstration of the practical capacity of predicate logic to define and combine data to give more abstract properties through which one can reason about machine state, but at some cost in confusing state transition processes with inference itself (difference between marked transition and inferred transition). Dynamic logic, [Pratt 1976], is rare in providing reasoning about the effect of programs on states of affairs, but in its primitive form it lacks process abstraction, and has no seriously executable form as a programming system. Fischer and Ladner, [Fischer and Ladner 1979], extend Hennessy-Milner Logic to propositional dynamic logic, PDL.

Extending state-based methods with temporal logic and replacing predicate calculus has suffered from lack of expressiveness or increased logical complexity. [Pnueli 1979] introduced a temporal logic for reasoning about concurrent programs with single modality "*forever*" but which is not expressive enough to completely describe programs. More expressive logics emerged e.g. "*next*", "*until*", "*since*". [Lamport 1994] introduces a temporal logic of actions for reasoning about algorithms and actions rather than about the temporal system. They consider verification of a program and its properties like safety, liveness and fairness using proof lattices, [Owicki and Lamport 1982]. While their logic is adequate for describing and proving

simple safety and liveness properties of a single program, there are useful properties that it cannot express. It does not support proofs that one program implements another and it does not permit a simple compositional semantics of programs, [Lamport 1994]. [Baldoni et al. 1998] propose a modal approach for reasoning about dynamic domains with a logic programming setting. They however do not define operators for sequential composition, non-deterministic choice and iteration of actions.

3.3.1 Situation Calculus

The basic concepts in situation calculus, [McCarthy and Hayes 1969], are *situations* and *actions*. A situation is a period of time during which a certain set of properties or list of propositions hold. McCarthy defines a situation as the complete state of affairs at some instant of time. Actions cause state transitions. Situation calculus is often used for describing how actions and other events affect the world. For example, a robot's state of mind may be regarded as a component of a situation and how mental events give rise to new situations are described. If an action *A* occurs in a situation *S*, a new situation *result*(*A*, *S*) becomes valid. The fluent *Holds*(*P*, *S*) means that proposition *P* is true in situation *S*. The fluents *initiates*(*A*,*S*,*P*) and *terminates*(*A*,*S*,*P*) respectively denote that action *A* in situation *S* initiates or terminates the property or proposition *P*. Frame axioms for these fluents, and others such as *occurs*, *changes*, *succeeds*, *happens*, *elaborates* are given in [McCarthy and Hayes 1969]. Situation calculus gives rise to the frame problem. There is a need to enforce that during an action, all fluents that the action does not affect should stay the same after the action. Thus a large number of axioms have to be added to the theory. $m \times n$ axioms are needed for *m* actions and approximately *n* fluents. This requirement for a large number of frame axioms in order to prove that most things stay the same as actions are performed is known as the *frame problem*.

The frame problem has been interpreted in a variety of ways yielding related problems such as the *persistence* problem, the *qualification* problem and the *ramification* problem. See [Shahanan 1997a] for a description of these problems. Monotonic solutions by Davis [1990] and Reiter [1991] allow known concurrent events but depend on strong assumptions. Such monotonic approaches do not solve the frame problem because they still use frame axioms. In nonmonotonic logics, in

contrast to classical logic, a conclusion may be retracted from the set of conclusions as new information or assumptions are gained or made. Circumscription, [McCarthy 1980], is a form of nonmonotonic logic where the extension of a predicate is restricted as much as possible. McCarthy proposes that if a fluent holds in a particular situation and an event occurs that is not abnormal with respect to this fluent, then the fluent will still be true in the situation resulting from performing that event. A maximum of facts will persist when minimising abnormality. McCarthy's solution does not work for concurrent actions and succumbs to the Yale Shooting Problem, [Hanks and McDermott 1986]. Solutions that aim to solve the frame problem by only considering the situation calculus are flawed because they focus on a specific version of the problem. The situation calculus has strong assumptions, such as lack of concurrent actions, where concurrency contributes to the ramification problem. McCarthy and Costello [1998] describe a new version of the situation calculus which allows concurrent events and information update through narratives. Events are explicitly dealt with through narratives. However this does not remove the need for frame axioms and an action may unexpectedly be undone by another action, as in the Yale shooting problem. Thus, in situation calculus, it is hard to express non-determinism and that no other events have occurred that might cause a precondition of an event in the narrative to fail.

The protocol for a negotiation does not involve knowledge about the rest of the world so a finite number of axioms are needed for specifying the states and actions in a negotiation. A protocol can use the relation between states and sub-states to ensure that only the current state of a negotiation is true while all other non-parent states are false.

3.3.2 Event Calculus

Another approach is the *event calculus*, [Kowalski and Sergot 1986; Kakas and Miller 1997], where time is explicitly represented for reasoning about actions. The event calculus models how the truth-value of a predicate changes because of events that occur at certain times, where time is either discrete or continuous. The event calculus is based on a temporal representation and does not adapt to other situation-calculus like systems. Planning in the situation calculus can be done by a constructive proof of the existence of a situation, whereas in the event calculus the planning can be

performed inductively or abductively. Two criticisms of the event calculus are its linear time formalism and semi-decidability. No point in time has two alternative successor states which are temporally unrelated to each other and the future never evolves into two different paths. Branching time assumes the potential evolution of time forms in a tree-like structure where the vertices of such a tree can be called *states* or points in time.

3.3.3 Modal Logic

Modal logic is the logic of necessity and possibility. Kripke models, used in the semantics of modal logic, are essentially relational structures where the worlds or states are nodes, propositions are unary relations and modalities are binary relations between states. *Necessity* is what is true at every possible world and *possibility* is what is true at some possible world. The syntax of propositional modal logic is as follows:

Atomic formulae: $p \in \phi$

Formulae: $A \in Fma(\phi)$

$A ::= p \mid \bot \mid A_1 \rightarrow A_2 \mid \Box A$

Let ϕ be a denumerable set of atomic formulae with typical member propositions, denoted by p. The set of all propositional formulae generated from ϕ is obtained from the function $Fma(\phi)$. The formula $\Box A$ indicates necessity in that the formula A holds in all possible worlds. Other connectives include the propositional logic operators (negation, disjunction, conjunction, equivalence, True) and $\Diamond$. The formula $\Diamond A$ is equivalent to $\neg \Box \neg A$. If the formula A is possible, then it is not the case that $\neg A$ holds in all accessible worlds.

A Kripke model $\mathcal{M}$ is a triple $(W, \{R_\alpha \mid \alpha \in MOD\}, V)$, where W is a non-empty set of states or worlds, R_α is a binary relation on W with modality α, and V is a valuation function with the set of propositions as domain and range the powerset of W. V indicates at which states each propositional symbol is true. Propositional modal representations are computable while first-order logic is undecidable.

The main defect of Kripke modal systems is the inability to explicitly access individual worlds using modal syntax. Although the semantics allow reference to individual worlds, the world's individuality is lost on the syntactical side. [Blackburn 2000] shows the asymmetry in modal logic and provides an overview of hybrid logic as a remedy. He argues that because of the inability to express individual states, modal logic is not an adequate representation formalism and it is difficult to devise usable modal reasoning systems. For example the notation *Hold*(*P*,*i*) in interval logic, [Allen 1984], meaning the property *P* holds at the interval *i*, is not represented in the modal logic of intervals, [Halpern and Shoham 1991]. The consequence is the existence of semantically equivalent Kripke models which are not isomorphic, since identity of states is not guaranteed. Some domains need to deal with states explicitly, to name them, reason about their identity and reason about the transitions that are possible between them.

3.3.4 Multi-Modal Logic - Propositional Dynamic Logic (PDL)

Dynamic logic, [Pratt 1976], defines formal systems for reasoning about actions, from a before and after point of view. Dynamic logic associates each command α of a programming language with a modal connective $[\alpha]$ and the formula $[\alpha]A$ is read as "*A* holds after α terminates". This yields a multi-modal language with a set of modal connectives indexed by the set of programs. (See Golblatt [1992] for the full specification and properties of PDL).

PDL allows the properties of complex programs to be expressed in terms of their constituent programs through modal connectives. Let ϕ be a denumerable set of atomic formulae with typical member propositions, denoted by *p*. The set of all propositional formulae generated from ϕ is obtained from the function $Fma(\phi)$. A non-atomic formula includes logical connectives. Let Π be a set of atomic programs from which other programs can be generated. Programs can be expressed in the same way as formulae. The types and syntax of PDL formulas and programs are as follows:

Atomic formulae: $p \in \phi$

Atomic programs: $\pi \in \Pi$

Formulae: $A \in Fma(\phi, \Pi)$

Programs: $\alpha \in Prog(\phi, \Pi)$

$A ::= p \mid \bot \mid A_1 \rightarrow A_2 \mid [\alpha] A$

$\alpha ::= \pi \mid \alpha_1 ; \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha^* \mid \mathrm{A}?$

The program $\alpha_1; \alpha_2$ means do program α_1 and then α_2. The program $\alpha_1 \cup \alpha_2$ is read as do either α_1 or α_2 non-deterministically. The program α^* means repeat α some finite number of times. The program $A?$ tests A and continues if A is *true*, otherwise fails.

3.3.5 Hybrid Logics

Hybrid logic, [Passy and Tinchev 1991, Blackburn 2000] enriches the syntax of modal logic to fit its semantics by incorporating reference to states. Hybrid logic restricts propositional variables to be interpreted as true at exactly one state, obtaining equality between states. Each name nominates exactly one state and each state has at least one name.

Passy and Tinchev [1991] specify CPDL (combinatory PDL) and add a new type to modal logic for names, forming a 4-tuple model: (W, R, χ, V). They define 3 countable infinite and pairwise disjoint alphabets - Σ denotes the set of names, ϕ_0 denotes the set of atomic propositions and Π_0 denotes the set of atomic programs. The set $\Sigma \cup \phi$ contain formulas. The model for CPDL is $\mathrm{M} = (W, R, \chi, V)$. W is a non-empty set of states or worlds. R: $\Pi \rightarrow \mathrm{P}(W^2)$ is the accessibility relationship according to the modality Π. The valuation function V has as domain formulae (including names) and range the power set of W. χ is a surjective function with domain Σ and range W. $V(c) = \{\chi(c)\}$, $c \in \Sigma$. $\chi(c)$ returns the unique world where the state c is true. The separability of $V(\neg c)$ from $V(c)$ enables syntactic treatment of a world's identity. See [Passy and Tinchev 1991] for axioms and rules of CPDL, extensions to it and analysis of its soundness, determinism, completeness and expressivenees.

Another approach for hybrid logic is by Blackburn [2000] who associates names to states by introducing *nominals* as a type of atomic formula. Nominals are true at exactly one point in any model. Thus a world or state is unique by virtue of its

nominal. For any nominal i, the symbol sequence $@_i$ is a satisfaction operator. Well-formed formulae can be propositions, modal formulae or $@_i \alpha$ where α is a formulae. $@_i \alpha$ is interpreted as α is satisfied at the unique point where i is true. Blackburn defines a hybrid model to be a triple, $(W, \{R_\alpha \mid \alpha \in MOD\}, V)$. V is a hybrid valuation function with domain propositions and nominals and range powerset of W. $V(i)$ is a singleton subset of W and the unique state in $V(i)$ is called the denotation of i. A world is identified by its unique state and named states can be reasoned about instead of only the current state.

Nominals	$i \in NOM$
$M, w \models i$	iff $V(i) = \{w\}$, $i \in$ NOM
$M, w \models @_i \varphi$	iff $M, w_1 \models \varphi$ and w_1 is the denotation of i

Nominals are atomic formulas and satisfaction operators are treated as normal modal operators. $@_i j$ asserts that the states named by nominals i and j are identical. $@_i R_\pi j$ indicates that the state named by j is an R_π-successor of the state named by i.

Hybrid logics are termed hybrid because they incorporate concepts of identity and reference from classical logic into modal logic. Modal logic and hybrid logic can be translated into classical logic. Areces, Blackburn and Marx [1999] show hybrid logics are of polynomial complexity and not more complex than multimodal logic. [Blackburn 2000] describes a system for hybrid logic by giving the rules, axioms and theorems in hybrid reasoning and its completeness. Examples are illustrated through tableaux systems. Quantified hybrid logic contain special notation for binding nominals to the current state, acting like ∃ and ∀ quantifiers, [Blackburn and Marx 2002]. This yields a hybrid logic with first-order expressivity. However this binding produces an undecidable satisfaction problem, [Areces et al 1999b].

3.3.6 Logics of Concurrency

Modal approaches for modeling actions can be found in [Giordano et al. 1997; Giacomo and Lenzerini 1995] with emphasis on concurrent actions. An example of a logic of concurrency is the *μ-calculus*, [Kozen 1983]. Propositional μ-calculus extends normal propositional modal logic through fixed-point operators. If α is a

formula and X is a variable, then $\mu X.\ \alpha$ and $\nu X.\ \alpha$ are formulae. If M is a Kripke structure, then $M, s \models \mu X.\alpha$ iff s is a member of the least fixed point of a certain function f: $2^s \rightarrow 2^s$. Similarly, $M, s \models \nu X.\alpha$ iff s is a member of the greatest fixed point of f. Recursion can then be expressed by these two fixed-point operators. The termination of two concurrent programs, α and β, can be expressed as $<\alpha>\ true \wedge <\beta>true$. Propositional μ-calculus covers the full expressive power of PDL with the same computational complexity and strictly subsumes CTL*, [Emmerson 1990]. Schild [2000] shows that two- dimensional BDI_{CTL*} can be collapsed into a single dimension and on certain conditions encoded as a conventional one-dimensional Kripke structure. He also proves that propositional μ-calculus with R-seriality is stronger in expressive power than BDL_{CTL*} and proposes a complete axiomatisation for the BDI theory using μ-calculus. Process logics and the μ-calculus reuse principles from propositional dynamic logic.

3.4 ANML as Extended PDL

Specifying a negotiation protocol involves specifying possible transitions from different states. There is no established notation to represent both the states and processes of an active agent, let alone a calculus for deciding why a particular negotiation should achieve the mutual goals. Hybrid logic and propositional dynamic logic seem better suited to represent theories for agent negotiation and their protocols. PDL provides reasoning about the effect of programs on states of affairs, allowing the relating of processes to goal states and fits naturally with a multi-modal theory of agent beliefs, goals and intentions. We have considered hybrid logics to represent protocols and negotiation so as to explicitly deal with states. However we found that unique states were not necessary for negotiation and therefore multi-modal logics are sufficient for representing protocols. For the scope of this thesis, extending PDL allows to deal with the explored issues of negotiation. For more expressiveness, ANML may be extended to incorporate the capabilities of hybrid logic.

Therefore we shall conceptually treat an agent as capable of atomic actions, each constituting a primitive process which may be combined in more complex ones. A suitably rich logic with action terms and variables appears capable of a practical reasoning system which can relate processes to goal states. The semantics for actions

and state transitions can be modelled through accessibility relations between possible worlds. Kripke semantics, [Kripke 1963], is applicable to a framework reasoning both about mental attitudes and time. Worlds can be viewed as program states and accessibility relations as non-deterministic programs for state transitions.

ANML, (Agent Negotiation Meta-Language), is proposed as a meta-language for building and validating interaction protocols and providing an abstract logical theory of a conversation, and thus negotiation. New or existing protocols can be expressed, verified and extended in ANML so as to analyse protocol properties. ANML is based on multi-modal logic; more precisely, it extends propositional dynamic logic (PDL) to express multi-agent interactions. ANML allows representation of complex actions and to reason about computational aspects such as properties of protocols. Programs in PDL are referred as processes in ANML. Representation of the states of nested negotiation processes is also supported. For example, the state s_1 holding after process α_1 is different from the state s_1 holding after execution of a different process α_2 (where α_1 possibly spawns α_2).

3.4.1 ANML for Specifying Protocols

The meta-language, ANML, aims to better represent protocols and constraints for state transitions than finite state machines. This chapter specifies the syntax and semantics for ANML, without needing to specify the semantics of any particular conversation, ACL, ontology or content language used. A protocol defines the structure of a conversation, that is the allowable sequence of exchanged messages. The messages themselves can be in any agent communication language. Thus, communicative acts in existing agent languages may be passed in ANML. It remains hard to standardise the semantics of the content of a conversation as misunderstandings and misinterpretations about them easily arise between agents. ANML thus maintains compatibility with ongoing standardisation efforts for an ACL. It addresses what can be standardised seamlessly without the danger of semantics mismatch in language and ontology between different agents.

ANML supports the specification of protocols such that they can be sharable and, in addition, exhibit testable and desirable properties. Well-defined ANML protocols can be used to help agents in reasoning about their goal seeking activities. In effect this

thesis provides a framework for representing, verifying and reasoning about negotiation. One of the main components of the framework is the ANML meta-language. The contributions of the framework may be enumerated as follows:

- specifying intuitive logical theories for agent protocols.
- verifying new and existing agent protocols.
- a meta-language for an agent to represent, reason and plan about the states and processes of negotiation, in the pursuit of its goal.
- proving safety and liveness properties and correctness of protocols.
- analysing consistency of common and joint beliefs in a MAS

This thesis applies and extends work on multi-modal logic and action logics to facilitate automated negotiation through interaction protocols.

3.5 Syntax of ANML

The syntax of ANML is an adaptation of the program logic described in [Pratt 1976]. ANML is specified in the same way as propositional dynamic logic in [Golblatt 1987]. ANML is defined over propositions, atomic processes and agents. Classical logic Boolean operators, list and set notations apply to ANML formulas e.g. $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, {}, [], $\neg$, $\perp$, T, $\cup$ and $\cap$. The types in ANML are as follows:

Propositions:	$p \in \phi$
Atomic processes:	$\varpi \in \omega$
An Agent:	$agent \in Agents$
Formulae:	$A \in Fma(\phi, \omega, Agents)$
Processes:	$\alpha \in Proc(\phi, \omega, Agents)$

Let ϕ be a denumerable set of propositions, with typical member denoted by p. ω is a set of atomic processes (actions), ϖ is an atomic process and is a member of ω. The set *Agents* is a set of all agents, with typical member *agent* as an agent. The set $Proc(\phi, \omega, Agents)$ is the set of all complex processes that can be generated from propositions, atomic processes and agents through ANML connectors. A typical

process is α, which is a member of *Proc*(ϕ, ω, *Agents*). The set *Fma*(ϕ, ω, *Agents*) is the set of all formulae which can be generated from atomic processes, propositions and agents. A formula, *A*, represents the relation between propositions, processes and agents. *Proc* and *Fma* are functions from which processes and formulae can be generated respectively.

The syntax and constructors of ANML is specified in Backus Naur Form, (BNF), [Naur 1960]. '::=' means "is defined as" in production rules and '|' means "or". ANML is defined over the above types as follows:

An agent:	*oneAg* ::=	*agent* | *agent*: *role*
Sets of agents:	*A_group* ::=	{ } | {*oneAg*} | $A\_group_1 \cup A\_group_2$
Groups of agents:	*Agent_group* ::=	*oneAg* | *A_group*

Set of states: *State-set* ::= {*A*} | $A \cup State\text{-}set_1$

Formulae:

$A ::= p \mid \bot \mid \top \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \neg A \mid A_1 \rightarrow A_2 \mid A_1 \leftrightarrow A_2 \mid A(Agent\_group) \mid [\alpha]\, A$
$\mid <\alpha>\, A \mid \alpha_1 :: \alpha_2 \mid none\text{-}of\,(State\text{-}set) \mid one\text{-}of(State\text{-}set)$

Processes

$\alpha ::= \varpi \mid \alpha_1 ; \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha^* \mid A? \mid \alpha? \mid Agent\_group \,.\, \alpha \mid null \mid abort$

An agent group, *Agent_group*, is one agent or a set of agents, where an agent may be typed with a role. An agent's name and its role identify a particular agent. A role is an informal classification for agents where an agent may be taken as an instance of a role. An example of a group of two agents is ({*roger*:*retailer*} ∪ {*bill*:*buyer*}), where *roger* and *bill* are agent identifiers and *retailer* and *buyer* are role identifiers respectively. The process ({*roger*:*retailer* , *bill*:*buyer*}).*transacting* denotes the two agents involved in the *transacting* process. Therefore this is a two-level naming structure for an agent.

The state of a process is a formula over propositions, processes and agents. A state can be parameterised by an agent group as in the formula *A*(*Agent_group*) e.g. *offered*({*roger* , *bill*}). A formula may give a hierarchical relation between states.

For example *requested* → *open* means that *requested* is a sub-state of *open*. The formula *A* holding after executing a process α is represented as the formula $[\alpha]A$ e.g. [*offer*]*offered*. A negotiation state is considered as the propositions holding in that world. The formula $(\alpha_1 :: \alpha_2)$ denotes that process α_1 is constrained to be of the same type as α_2. That is, all the states and transitions allowed in process α_1 can also be inferred from α_2 (e.g. *E-bay-auction* :: *English-auction*). The process α_1 is an instance of the same class of process as α_2, but involving a different group of agents and subject of interaction.

A complex process such as negotiation may be expressed in terms of its sub-processes using the composition operator ";" or PDL connectors. For example, a negotiation process can be decomposed into the sub-processes of *browsing*, *bargaining* and *paying*. The state of a process depends on which of its sub-processes has been executed. In addition to testing atomic states (PDL allows tests on atomic states only), the process *A?* also applies when *A* is a compound formula and therefore in ANML, the test operator is used in its full generality over formulae. The executor of a process and that process are separated with a full stop (e.g. *r:retailer.display* means *retailer r* executes the *display* process). A *null* process represents no execution while an *abort* process results in a failed state.

3.6 Semantics of ANML

An agent is associated with processes by prefixing a process with an agent in the same way an object is suffixed by its methods. Usually the agent type is omitted and a joint process between two parties is denoted by the set of the two parties performing the process as in $(\{c, r\})$.*shopping*. A process may be decomposed into a sequence of sub-processes each possibly coupled with the agent or agents executing that sub-process. An action is an atomic process. The process denoted by *a*;*b* is composed of *a* followed by *b* in sequence, *a** denotes zero or more iterations of process *a* and a^+, one or more iterations of process *a*. For example, a shopping process between two agents *r* and *c* is composed of a sequence of sub-processes executed by the two agents: $\{r,c\}$. *shopping* = *r*:*retailer.display*; *c*:*buyer.browse*$^+$; *c*:*buyer.choose**. A state test operator '?' allows sequential composition to follow only if successful. For

example *c.browse*?; *c.choose* is the process *c.choose* if *c.browse* succeeds, otherwise it fails. We remark at this stage that conventional program-like if … then…, repeat … until … commands could be more intuitive than the primitive test, union and iteration operators, ?, ∪, *, $^+$, borrowed from dynamic logic for task composition, but there are tougher criteria in relating successful task composition to the goals and constraints of the agents.

3.6.1 Standard Models for ANML

Execution of an atomic or complex process in an initial world validates some resulting formula in an accessible world. A state transition is a change of state through performing a process. In multi-modal logic, each modal connective is expressed as an accessibility relation in a model. See Appendix A for a description on specifying the semantics of modal, multi-modal and propositional dynamic logic using standard Kripke models.

In ANML, a binary accessibility relation, R_α, applies to a set of worlds for a process α. The relation R_α reflects the intended meanings of the execution of a process α. In a model $M = (W, R_\alpha, V)$, the accessibility relation R_α is defined as the set $R_\alpha \subseteq (W \times W)$ i.e. R_α is an accessibility relation between w_1 and w_2, expressed as $w_1 R_\alpha w_2$, where w_2 and w_1 are worlds in W. w_2 is reached from the world w_1 via the execution of process α.

A Kripke model for ANML is $M = (W, \{R_\alpha: \alpha \in Proc(\phi, \omega, Agents)\}, V)$

$M = (W, \{R_\varpi, R_{null}, R_{abort}, R_{\alpha;\beta}, R_{\alpha\cup\beta}, R_{\alpha?}, R_\alpha^+, R_{A?}, R_{Ag\_group.\alpha}\}, V)$

where $\varpi \in \omega$, $\alpha \in Proc(\phi, \omega, Agents)$, $\beta \in Proc(\phi, \omega, Agents)$, $A \in Fma(\phi, \omega, Agents)$, $\Gamma \in protocols$, $Ag\_group : Agent\_group$

W is the set of worlds in model M. V is an evaluation function with domain propositions and range the powerset of worlds.

$$V: \phi \rightarrow 2^{Worlds}$$

The function V represents an assignment of sets of possible worlds to propositions. It tells us at which worlds (if any) a proposition is *true*. $V(p)$ is the set of worlds where

atomic formula p holds, as an interpretation of the atoms in the model. The Boolean connectives in propositional logic, $\wedge$, $\vee$, $\neg$, $\leftrightarrow$, $\rightarrow$, apply in ANML formulae:

$M, w \models p$ iff $w \in V(p)$, $p \in \phi$

$M, w \models A \wedge B$ iff $M, w \models A$ and $M, w \models B$

$M, w \models A \vee B$ iff $M, w \models A$ or $M, w \models B$

$M, w \models A \rightarrow B$ iff $M, w \not\models A$ or $M, w \models B$

$M, w \models \neg A$ iff $M, w \not\models A$

$M, w \models A \leftrightarrow B$ iff $M, w \models A \rightarrow B$ and $M, w \models B \rightarrow A$

ANML Modalities

The modal operators in ANML are as those in PDL, $[\alpha]$ and $<\alpha>$, which respectively mean true in all accessible worlds after execution of α and possibly true in an accessible world after performing α. In ANML, the modality $[\alpha]A$ is read as the formula $[\alpha]A$ is valid in w_1 iff after executing process α, the formula A holds in every world accessible to w_1. The formula $<\alpha>A$ is read as valid in w_1 iff after executing process α, the formula A holds in some world accessible to w_1. In w_1, the consequence of executing process α is known. The modality associated with α is true in w_1 if and only if the results of carrying out α holds in a set of worlds (depending on the modality) accessible to w_1.

$M, w \models [\alpha] A$ iff $\forall w_1$ ($w R_\alpha w_1$ implies that $M, w_1 \models A$)

$M, w \models <\alpha> A$ iff $\exists w_1$ ($w R_\alpha w_1$ and $M, w_1 \models A$)

where $w, w_1 \in W$, $\alpha \in Proc(\phi, \omega, Agents)$, $A \in Fma(\phi, \omega, Agents)$

3.6.2 Formulas

Well-formed formulas in ANML are made up of propositions, processes, agents, Boolean and ANML connectors. The evaluation function in a Kripke model for ANML is an assertion of the propositions that are true at a world. Proposition p is true in model M and world w iff w is an element of the set of worlds where p holds, given by $V(p)$.

$M, w \models p$ iff $w \in V(p)$, $p \in \phi$

The state, *A*, of a process is the set of worlds where *A* holds. The state, *A*, of a process such as negotiation is the set of worlds where *A* holds and all participants in the negotiation believes the formulae in *A* to be *true*.

State of a process parameterised with an agent or a group of agents

Execution of a process by an agent or a group of agents renders a resulting formula to be *true* in a world. The actions or messages between a group trigger successive states in a negotiation process. The formula *A*(*Agent_group*) is read as the formula *A* parameterised with a group of agents *Agent_group*. For example, from a protocol and the state *offered*(*Agent_Smith*), it can be inferred that *Agent_Smith* previously successfully performed a process that rendered *offered*(*Agent_Smith*) *true*. The meaning of a state parameterised by an agent depends on the rules and parameterisation of the protocol it occurs in. For example, the parameterisation of state A_2 in a protocol may be $A_1(Y) \rightarrow [X.\alpha]A_2(X)$.

$M, w \models A(X)$ iff $M, w \models A$ and $X \in Agent\_group$

ANML inherits the axioms of a normal modal system and the underlying modal logics are decidable. In addition, we have the axiom $(A(X) \rightarrow A)$ (e.g. *offered*(*X*) → *offered*). We assume that groups of agents in an interaction are finite sets and therefore our formalism does not embody mixed quantification.

States and sub-states

The state of a process is given by the truth-value of a formula. The relation between states is inferred from the theory of a protocol. The relation between states is expressed as a hierarchy using Boolean operators. The formula $A_1 \rightarrow A_2$ can be read as A_1 is a sub-state of A_2 or A_2 is a parent state of A_1. For example, in the formula *auctioned*(*X*) → *posted*(*X*), the state *auctioned*(*X*) is a sub-state of *posted*(*X*). In an auction, this formula implies that an item cannot be *auctioned*(*X*) without being *posted*(*X*). All actions that are possible from an *auctioned*(*X*) state are also possible

from the *posted*(*X*) state. However an item can be *posted* without being *auctioned*. There may be a process possible from *posted*(*X*) without being so from *auctioned*(*X*).

The formula $A_1 \leftrightarrow A_2$ is read as a process cannot be in the state A_1 without being in state A_2 and vice versa. Equivalencies enforce that a process cannot be in a sub-state without being in its parent state and vice versa. In this way it is expressed that a negotiation can only be in some sub-state. For example, the formula *closed* $\leftrightarrow$ (*agreed* $\vee$ *rejected*) $\wedge \neg$(*agreed* $\wedge$ *rejected*) implies that the states *agreed* and *rejected* are sub-states of *closed*. The state of a process cannot be *closed* without being in either *agreed* or *rejected* (but not both) and it cannot be *agreed* or *rejected* without being *closed*. Equivalency between states ensures that all actions possible from a sub-state are also possible from its parent state and vice versa.

Knowing only the truth-value of a parent state reflects partial information about the state of a process e.g. knowing that a negotiation is *closed* without knowing if it is *agreed* or *rejected*.

none-of and one-of

The operators *none-of* and *one-of* are used to express exclusivity between states. For example a process cannot be *open* and *closed* at the same time. The predicate *none-of* takes a set of formulae and returns *true* if all its elements are false. The predicate *one-of* returns *true* if exactly one of the elements in its given set is *true*.

$M, w \vDash$ *none-of*(*State-set*) iff $\forall A$ ($A \in$ *State-set* implies $M, w \nvDash A$)

$M, w \vDash$ *one-of*(*State-set*) iff $\exists A_1$ ($A_1 \in$ *State-set*$_1$ and $M, w \vDash A_1$ and

$\neg \exists A_2$ ($A_2 \in$ *State-set*$_1$ and $M, w \vDash A_2$ and $\neg (A_1 \leftrightarrow A_2)$))

For example *none-of* ({ A, B, C }) $\leftrightarrow \neg A \wedge \neg B \wedge \neg C$, while *one-of* ({ A, B, C}) $\leftrightarrow$ ($A \wedge \neg B \wedge \neg C$) $\vee$ ($\neg A \wedge B \wedge \neg C$) $\vee$ ($\neg A \wedge \neg B \wedge C$). The formula *closed* $\leftrightarrow$ (*agreed* $\vee$ *rejected*) $\wedge \neg$(*agreed* $\wedge$ *rejected*) can now be written as *closed* $\leftrightarrow$ *one-of* ({*agreed*, *rejected*}).

State transitions

A protocol expresses the actions leading to changes of state. In ANML, state transitions are represented as the formula $A_1 \leftrightarrow [\alpha]A_2 \vee [\rho]A_3$. The initial state is A_1 in world w_1 and execution of process α leads to A_2 holding in all worlds accessible to w_1 through the accessibility relation R_α. Execution of process ρ leads to A_3 holding in all worlds accessible to w_1 through the accessibility relation R_ρ. The relation between states in a protocol expresses whether states A_1, A_2 and A_3 can co-exist. Hence, the formula $A_1 \leftrightarrow [\alpha]A_2$ represents a state transition from state A_1 to state A_2 via execution of process α. A_1 is the previous state of A_2 and A_2 is viewed as the next state of A_1. If α is the *null* process, then the equivalency is a relation between parent and sub-states.

Process Instances

The semantics of $(\alpha_1 :: \alpha_2)$ is given by:

$M, w \vDash (\alpha_1 :: \alpha_2)$ iff $R_{\alpha_1} \subseteq R_{\alpha_2}$

All the worlds obtained through execution of process α_1 are elements of the set of worlds possible through performing α_2.

3.6.3 Processes

Atomic Process

A binary relation, R_ϖ, is assigned to each atomic process ϖ, where $\varpi \in \omega$. The mapping of relation R_ϖ from world w_1 to world w_2, (w_1, $w_2 \in W$), is given the same meaning as executing atomic process ϖ in world w_1 resulting in the world w_2. From $w_1 R_\varpi w_2$ and w_2, $w_1 \in W$, it can be inferred that w_2 is accessible to w_1 via R_ϖ and the result of executing ϖ holds in w_2.

Complex Process

A complex process is executed by performing its sub-processes until its atomic sub-processes are executed. Execution of an atomic or complex process has a mapping from an initial world to a final world where the consequence of its execution is true. The binary relation R_α is a relation on worlds W for the process α, R_α: $(W \times W)$. Process α is executed in $w_1 \in W$ to give rise to another $w_2 \in W$ where the results of

the execution hold. The world w_2 is possible relative to w_1 and the accessibility relationship between these two worlds is the relation R_α.

For a complex process, possible paths through intermediate worlds lie between an initial and a final world. These paths represent partial execution of that process. The relation R_α reflects the intended meaning of the process α and is inductively defined in terms of its sub-relations, which are partial paths. A model M for ANML is standard if it satisfies the conditions of defining R_α in terms of its sub-relations – composition, alternation, etc. ANML has a number of constructs, which resemble those of PDL, for process management. In this way, a uniquely determined standard model for ANML is obtained by inductively defining relation R_α for non-atomic processes α, in terms of binary relations, R_β, where β are the sub-processes of α.

An agent executing a process

An agent or a group of agents, *Agent_group*, performing a process, α, is represented by prefixing that process with the agent group as in $Agent\_group.\alpha$. The relation $R_{Agent\_group.\alpha}$ maps world w_1 to world w_2 through agent or group *Agent_group* executing process α. The set of worlds in the image of relation $R_{Agent\_group.\alpha}$ is a subset of the set of worlds in the image of R_α

$$R_{Agent\_group.\alpha} \subseteq R_\alpha$$

In the case of two processes executed by two groups of agents, i.e. $Agent\_group1.\alpha$ and $Agent\_group2.\alpha$, where $Agent\_group1 \subset Agent\_group2$, if the process α is the same instance when being executed by the two groups, then $R_{Agent\_group1.\alpha} = R_{Agent\_group2.\alpha}$. Since *Agent_group1* is sufficient to perform that instance of α, the results of that process would be the same regardless of the additional agents in *Agent_group2*. On the other hand, if the two groups are performing different instances of process α, then no relation between the two processes $Agent\_group1.\alpha$ and $Agent\_group2.\alpha$ are derivable before execution.

Process Composition

An overall complex process is the execution of all its sub-processes. A process P may consist of a sequence of sub-processes, where performing P involves performing its sub-processes sequentially. The process $\alpha_1;\alpha_2$ is composed of two sub-processes – process α_1 followed by process α_2. For example, the complex process *agent_smith.offer*; *agent_neo.reject* is read as after *agent_smith* makes an offer, *agent_neo* performs a rejection. The relation $R_{\alpha;\beta}$ for a sequential process $(\alpha; \beta)$ can be inductively defined as relational composition of the relations for each sub-process α and β.

$$R_{\alpha;\beta} = R_\alpha \circ R_\beta = \{(s, t) \in Worlds: \exists u(s\ R_\alpha u\ \&\ u\ R_\beta\ t)\}$$

$$[\alpha; \beta]\ C \leftrightarrow [\alpha][\beta]C$$

where $C \in Fma(\phi, \omega, Agents)$ and $\alpha,\ \beta \in Proc(\phi, \omega, Agents)$,

A sequential process execution is a path along the worlds where intermediary states hold after each sub-process. $\alpha; \beta$ is a path from the source world before executing α to the final world after executing β.

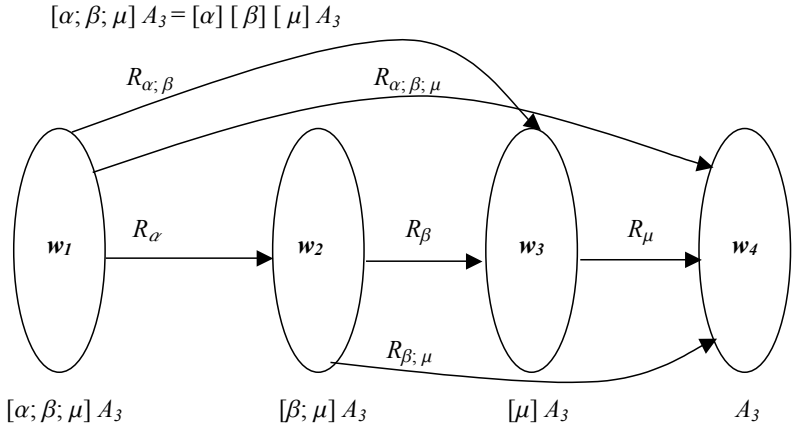


Figure 3.1 Process Sequencing with 3 sub-processes, $[\alpha; \beta; \mu]\ A_3$

In figure Figure 3.1, processes α, β and μ (relations R_α, R_β and R_μ) can themselves be expressed in terms of their sub-processes (relations).

Example: Two agents X and Y are engaged in the complex process *bargain* with a resulting state *agreed*(Y) if successful.

$[\{X,Y\}.bargain]\ agreed(Y) \equiv [X.request;\ Y.request;X.offer;Y.agree]\ agreed(Y)$

After the sub-processes *X.request*, *Y.request*, *X.offer* and *Y.agree*, the states *requested*(*X*), *requested*(*Y*), *offered*(*Y*) and *agreed*(*Y*) hold respectively. The state *requested*(*X*) is the source state for the process *Y.request*.

Process Alternation

The process $\alpha_1 \cup \alpha_2$ denotes executing either of the two processes non-deterministically, with the same resulting formula. This gives non-deterministic choice between a number of processes. The corresponding accessibility relation $R_{\alpha\cup\beta}$ is equal to the union of the sub-process relations.

$$R_{\alpha\cup\beta} = R_\alpha \cup R_\beta$$

$$[\alpha \cup \beta]\, A = [\alpha]\, A \wedge [\beta]\, A, \qquad A \in Fma(\phi, \omega, Agents)$$

Example: Agent *Neo* can execute the process *eat_meal* with resulting state *not_hungry*. The *eat_meal* process consists of either eating an Indian or Chinese meal.

$$[Neo.eat\_meal]\ not\_hungry(Neo) \equiv$$
$$[Neo.eat\_chinese \cup Neo.eat\_indian]\ not\_hungry(Neo) \equiv$$
$$[Neo.eat\_chinese]\ not\_hungry\ (Neo) \wedge [Neo.eat\_indian]\ not\_hungry(Neo)$$

Testing a Formula

Testing a state, *A*?, is treated as a process. A formula can be tested to see if it holds in a world. Such a test may be necessary before carrying on with another process, e.g. the state *offered* triggered by an *offer* must be tested before agreeing to the *offer*. It can be checked that intermediate states hold before executing a sub-process by the state test *A*?. Testing a state may be used as a condition and combined with iteration constructs to define loops. Testing a formula in a world returns the same world where that formula is *true* or *false* depending on the success of the test.

$$R_{A?} = \{(w, w) : M, w \models A\}$$

Example: The door must be *open* for agent *X* to go to the dining room. There are two processes here: 1) testing the door is *open* followed by, 2) agent *X* going to the dining room.

$X.dine = open?; X.goto\_diningroom$

Testing the success of a process

The success of a process execution is tested by checking whether its consequential end state holds.

$$R_{\alpha?} = \{(w_1, w_2) : (w_1, w_2) \in R_{\alpha}\},$$

In relation $w_1 R_{\alpha?} w_2$, the process $\alpha?$ succeeds if the process α succeeds.

Example: agent *X* must open the door in order to go to the dining room. On the condition that the door is successfully opened, that agent will be able to eat.

$X.dine = X.open?; X.eat$

Process Iteration

A process can be repeatedly executed, such as when polling for a resource. Iteration can be combined with testing to construct *while* and *repeat-until* loops. A relation for representing repetition of a process zero or more times is defined in terms of relational composition. The process α^* is zero or more iterations of the process α, whereas the process α^n signifies process α is repeated *n* times where $n \geq 0$. The process α^+ is read as iteration of process α one or more times. The '$^+$' construct can be expressed in terms of the '*' construct.

$$R_{\alpha^*} = R_{\alpha\,;\,\alpha^*} = (R_{\alpha})^*$$
$$\alpha^*;\gamma \leftrightarrow \gamma \vee \alpha;\gamma \vee (\alpha^*);\gamma$$
$$[\alpha^*]\,A \leftrightarrow [\alpha][\alpha^*]\,A$$

Example: Agent *Morpheus* repeatedly executes the process of *shopping* if the shop is open.

(*open*?; *Morpheus.shopping*)* =

(*open*?; *Morpheus.shopping***)** ; (*open*?; *Morpheus.shopping*)*

The process α^+ captures the condition of *do-while* loops. It is a variant of the above iteration.

$R_{\alpha+} = R_{\alpha;\, \alpha^*} = R_\alpha \circ R_{\alpha^*} = R_\alpha \circ (R_\alpha)^*$

$[\alpha^+]\, A = [\alpha]; [\alpha^*]\, A$

Iteration of a process can also be bounded with inequalities or made finite if superscripted with a positive integer.

Example: $\alpha^2 = \alpha; \alpha$

$\alpha^{<3} = null \cup \alpha \cup \alpha;\alpha$

Agent *Morpheus* eats zero or more times is represented as *Morpheus*.eat*

Agent *Morpheus* eats at least once is represented as *Morpheus*.eat$^+$

Agent *Morpheus* eats 10 times sequentially is *Morpheus.eat*^{10}

Agent *Morpheus.eat*^{10} = *Morpheus.eat*; (*Morpheus.eat*)^9

Agent *Morpheus* eats at least 3 times is represented as *Morpheus*.eat$^{\geq 3}$

Agent *Morpheus* eats less than 7 times is represented as *Morpheus*.eat$^{< 7}$

The *null* process

The *null* process does not change the world. R_{null} is the identity relation.

null is $\top$?

$R_{null} = \{(w,w): w : W\}$

The *abort* process

abort is an abnormal process that leads to an inconsistent and *failed* state.

abort is $\bot$? where R_{abort} is the null set ϕ.

3.7 Axioms and Inference Rules in ANML

Let Σ be a normal logic system in $Fma(\phi, \omega, Agents)$. The axioms and inference rules of classical propositional logic and propositional dynamic logic hold in ANML, giving a Hilbert-style deductive system for ANML. As shown in section 3.7.3, both the modal axiom Df$\Diamond$ and rule of inference RK, [Chellas 1980], hold in ANML. Because ANML is an extension of PDL and hence multi-modal logic, all the axioms

and inference rules of a normal system in modal logic hold in ANML. See Appendix A for axioms and rules for a normal modal system. This section and section 3.9 provide the additional axioms and rules that apply to ANML due to its extensions and additional connectives.

3.7.1 Precedence of ANML Operators

The operators between processes have higher precedence than the operators between formulae. The precedence of the operators in ANML in increasing order is as follows:

$\wedge, \vee, \rightarrow, \leftrightarrow$	least binding
$;, \cap, \cup$	
$>>$	
$*, ^{+}, ?,$	
$\neg$	
$\cdot$	
$:$	most binding

Table 3.1 Precedence of ANML operators

3.7.2 Constructs from ANML Connectives

The formula α^{n+1} signifies executing process α sequentially $n+1$ times, that is carrying out the process α, n times and one more time.

$$\alpha^{n+1} = (\alpha^{n}; \alpha) = (\alpha ; \alpha^{n}).$$

α^{0} is the *null* process. Literally it means performing process α zero times, which is not at all.

There is an intermediate state when executing two processes sequentially.

$$[\alpha; \beta]\, C \leftrightarrow ([\alpha]B \wedge (B \rightarrow [\beta]\, C)) \leftrightarrow ([\alpha]; B?; [\beta]\, C) \leftrightarrow [\alpha][\beta]C$$

There are other constructs that can be derived from ANML operators such as *if then else*, *while* or *repeat-until* loops. These procedural constructs do not contribute a significant degree of conciseness from what is already defined and are not particularly useful in expressing the logical theory of a protocol.

PDL Axioms, valid also in ANML

Composition	$[\alpha;\rho]\, A \leftrightarrow [\alpha][\rho]\, A$
Alternation	$[\alpha \cup \rho]\, A \leftrightarrow [\alpha]\, A \wedge [\rho]\, A$
Mix	$[\alpha^*]\, A \rightarrow A \wedge [\alpha][\alpha^*]\, A$
Ind.	$[\alpha^*]\, (A \rightarrow [\alpha]A) \rightarrow (A \rightarrow [\alpha^*]\, A)$
State Test	$[A?]B \leftrightarrow (A \rightarrow B)$

Additional axioms with respect to ANML connectors are:

Mix over $^+$	$[\alpha^+]\, A \rightarrow [\alpha][\alpha^*]\, A$
Ind. over $^+$	$[\alpha^+]\, (A \rightarrow [\alpha]A) \rightarrow ([\alpha]A \rightarrow [\alpha^+]\, A)$
Process Test	$\alpha?B \leftrightarrow [\alpha]B$
Process Imp. Test	$[\alpha?]B \leftrightarrow ([\alpha]A \wedge A \rightarrow B)$
Null Process	$[null]A \leftrightarrow A$
Abort Process	$[abort] \perp$

3.7.3 Axioms and Inference rules in ANML

Provided below are inference rules in ANML over $Fma(\phi, \omega, Agents)$, in addition to those in appendix A for a normal system of modal logic. Let formulae $A, B \in Fma(\phi, \omega, Agents)$, processes $\alpha, \beta, \rho \in Proc(\phi, \omega, Agents)$ and Σ be a normal logic system in $Fma(\phi, \omega, Agents)$.

- **Modal axiom Df◇ (definition of ◇)**

Df◇	$<\alpha>A \leftrightarrow \neg([\alpha]\neg A)$

The formula $\neg([\alpha]\, \neg A)$ is read as it is not the case that in every possible world, the formula $\neg A$ holds after executing process α. If α consists of sub-processes, then $<\alpha>A$ means there is a possible path α to a world where A holds. There is a possible path to state A after the execution of α iff not all executions of α lead to $\neg A$.

- **Rule of Inference RK**

RK $$\frac{(A_1 \wedge \ldots \wedge A_n) \rightarrow A}{([\alpha]A_1 \wedge \ldots \wedge [\alpha]A_n) \rightarrow [\alpha]A} \qquad (n \geq 0)$$

The formula $((A_1 \wedge \ldots. \wedge A_n) \rightarrow A)$ implies that the states A_1 to A_n are sub-states of A and all have to be valid in the same world for A to hold. This takes care of the situation where an action α_m may undo another α_p, since all the sub-states 1 to n must hold. If process α triggers all the sub-states of A, it brings about A too. In ANML, the rule of inference *RK* is weak since it assumes that α can trigger states A_1 to A_n. A stronger rule would include $([\alpha_1]\, A_1 \wedge \ldots. \wedge [\alpha_n]\, A_n)$ in the conclusion, which would need processes α_1 to α_n to be concurrent.

- **RK^{-1}**

RK^{-1} $$\frac{([\alpha]A_1 \wedge \ldots \wedge [\alpha]A_n) \rightarrow [\alpha]A}{(A_1 \wedge \ldots \wedge A_n) \rightarrow A} \qquad (n \geqslant 0)$$

The inference rule *RK^{-1}* indicates that if process α triggers states (A_1 to A_n) and state A, then it can be inferred that (A_1 to A_n) are sub-states of A. Again this rule may be extended to concurrent processes by substituting for α as different processes i.e. $([\alpha_1]\, A^1 \wedge \ldots. \wedge [\alpha_n]\, A^n)$.

- **Variant RK**

A variant of RK is if states A_1 to A_n are disjunctive sub-states of A i.e. if any one of the sub-states is true then the parent state A also holds. Then any processes, $(\alpha_1 \ldots. \alpha_n)$, triggering a sub-state also triggers the parent state A.

Variant RK $$\frac{(A_1 \vee \ldots \vee A_n) \rightarrow A}{([\alpha_1]A_1 \vee \ldots \vee [\alpha_p]A_n) \rightarrow [\alpha]A}$$

$(n,\ p \geqslant 0)$ and $(\alpha_1 \cup \ldots. \cup \alpha_p = \alpha)$

- **Sequential RK**

Let states A_1 to A_n be all mutually exclusive sub-states of A i.e. A_1 to A_n hold in parallel and are not sub-states of each other. The formula $[\alpha_i]A_i \wedge [\alpha_j]\, A_j \rightarrow ([\alpha_i ; \alpha_j]\, (A_i \wedge A_j))$ implies that process α_i does not undo the results of process α_j or vice versa. If these formulae hold then processes α_1 to α_n can be executed sequentially (and in any order) with the parent state A holding.

Sequential RK

$$\frac{(A_1 \wedge ... \wedge A_n) \rightarrow A, [\alpha_i]A_i \wedge ... \wedge [\alpha_j]A_{i+j} \rightarrow [\alpha_i;....;\alpha_{i+j}](A_i \wedge .... \wedge A_{i+j})}{([\alpha_1]A_1 \wedge ... \wedge [\alpha_n]A_n) \rightarrow [\alpha]A}$$

$(n \geqslant 0)$, ($n \geqslant i+j$, $k+m \geqslant i, k \geqslant 0$ and $(\alpha_k; \ldots. ; \alpha_{k+m} = \alpha)$

- **Variant RK^{-1}**

If processes α_1 to α_n triggers states A_1 to A_n respectively and all the processes triggers A, then A_1 to A_n are sub-states of A.

$$\text{Variant RK}^{-1} \quad \frac{([\alpha_1]A_1 \vee ... \vee [\alpha_n]A_n) \rightarrow [\alpha]A}{(A_1 \vee ... \vee A_n) \rightarrow A}$$

$(n \geqslant 0)$ and $(\alpha_1 \cup \ldots. \cup \alpha_n = \alpha)$

3.7.4 Examples of negotiation through multi-modal axioms and rules

The rules *RN*, *RM*, *RR* and *RE* follow from *RK* for $n = 0, 1, 2$ and applying equivalency respectively. See Appendix A for the rules of inference *RN*, *RM*, *RR* and *RE* in ANML. See also Appendix A for the ANML version of modal axioms *N*, *M*, *C*, *R* and *K* and their proofs. Here are some examples of how these rules and axioms are useful for reasoning about a negotiation.

RN: The inference rule *RN* states that if *A* is a tautology then *A* holds in all the worlds after the execution of any process α.
RM: If *A* implies *B* and process α triggers *A*, then, from *RM*, process α triggers *B*.
RR: If *A* and *B* are sub-states of *C*, then from rule *RR* a process α that triggers *A* and *B* also triggers the parent state *C*.

Example for rule *RM*: If a protocol allows the formulae *proposed* $\rightarrow$ *offered* and [*propose*]*proposed*, where the state *proposed* is triggered from making a proposal. Then making a proposal also triggers its parent state *offered*.
From *RM*, [*propose*] *proposed* $\rightarrow$ [*propose*] *offered* and thus [*propose*] *offered* can be inferred from the protocol.

Example for rule *RR*: Let a protocol contain the formula (*open* ∧ *bidded*) → *auctioning*, where *open* and *bidded* are sub-states of *auctioning*. Then if a process triggers either of the sub-states as in [*bid*]*open* ∧ [*bid*]*bidded*, then that process (*bid*) also triggers the parent state (*auctioning*) i.e. [*bid*]*auctioning*.

RE is the rule of inference between equivalent states and infers that the same process triggers equivalent states.

Example: Let *A* be the state *open* and *B* be its only sub-state *offered* according to a protocol, where *offered* ↔ *open*. If a process e.g. *offer* triggers the *offered* state as in [*offer*]*offered*, that process also triggers the *open* state, i.e. [*offer*]*open* and vice versa.

For states *A* and *B* to hold in the same world (as in $M, w \vDash (A \wedge B)$), either one state is a sub-state of the other or they are mutually exclusive states that can be valid at the same time. Axioms *M* and *C* (see appendix A) entails that a process that triggers a conjunction of states, triggers each state and vice versa.

Axiom *K*: If after executing process α, state *A* is a sub-state of *B*, then if process α triggers the state *A*, it also triggers its parent state *B*.
Example: ([*offer*] (*offered* → *open*)) → ([*offer*]*offered* → [*offer*] *open*)
Making an *offer* yields both an *offered* state and its parent state *open*.

3.7.5 Rules and Axioms for <α>

Appendix A gives the rules and axioms underlying the possibility modality <α> holding in a normal system for ANML. By the schema Df◇ <α>*A* is ¬([α] ¬*A*). <α>*A* has the modal meaning of possible worlds i.e. <α>*A* holds in world w_1 if after executing α, there is some world(s) accessible to w_1 where *A* holds. In the context of a graph or a negotiation, <α>*A* can be read as the possible path α to a goal state *A*.

- **RK◇, RN◇, RM◇, RR◇and RE ◇**

$$\text{RK}\Diamond \qquad \frac{A \rightarrow (A_1 \vee \ldots \vee A_n)}{< \alpha > A \rightarrow (< \alpha > A_1 \vee \ldots \vee < \alpha > A_n)} \qquad (n \geqslant 0)$$

Rule *RK*◇ expresses that if there is a path to a sub-state, then there is a possible path to at least one of its parent states.

Inference rules *RN*◇, *RM*◇, *RR*◇ follow from rule *RK*◇ for $n = 0$, 1, and 2 respectively. See appendix A for formal definitions of inference rules *RN*◇, *RM*◇, *RR*◇ *and RE*◇.

Rule *RN*◇: If $\neg A$ is a tautology then there is no possible path to *A*.
Rule *RM*◇: A possible path to a sub-state is a possible path to its parent state too.
Rule *RR*◇: A possible path to a sub-state with 2 parent states is a possible path to at least one of the parent states.
Rule *RE* ◇: If *A* and *B* are mutually dependent sub-state and parent state (or are the same state), then a possible path to *A* implies a path to *B* and vice versa.

- **Df** □

  Df □ $[\alpha] A \leftrightarrow \neg (<\alpha>\neg A)$

  Proof: using *PL*, Df ◇ and RE, similar to Df□ for modal logic.

If all paths lead to *A* then it is not possible to get to a world where $\neg A$ holds. This axiom is useful when analysing the properties of a protocol e.g. termination or agreement.

- **Axioms N◇, M◇,C◇and R◇**

See appendix A for axioms *N*◇, *M*◇, *C*◇and R◇ that apply in ANML.

- **Axiom K◇**

  K◇ $(\neg(<\alpha> A) \wedge <\alpha>B) \rightarrow <\alpha> (\neg A \wedge B)$

  Proof From $B \rightarrow (A \vee(\neg A \wedge B))$ and RR◇, similar to K◇ for modal logic.

If process α does not lead to state *A* but possibly to state *B*, then executing process α yields a possible world with $(\neg A \wedge B)$.

3.8 *D*, *T*, *B*, *4* and *5* Properties in ANML

A theory in ANML is characterised by the smallest normal system of modal logic *K*. The properties of *D*, *T*, *B*, *4* and *5* – serial, reflexive, symmetric, transitive and euclidean respectively – are analysed to see if they apply when applying ANML for negotiation. Let $M = (W, \{R_\alpha: \alpha \in Proc(\phi, \omega, Agents), V)$ be a model in ANML. This section shows that the relation R_α is not serial, reflexive, symmetric, transitive or euclidean in the domain of negotiation. Therefore we do not need ANML to exhibit any of the *D*, *T*, *B*, *4* and *5* properties. Let $w, w_1, w_2, \ldots w_n$ be worlds in W.

D. R_α is serial iff $\forall w\ \exists w_1\ (w\ R_\alpha\ w_1)$ does not hold in ANML

T. R_α is reflexive iff $\forall w\ (w\ R_\alpha\ w)$ does not hold in ANML

B. R_α is symmetric iff $\forall w, w_1\ (w\ R_\alpha\ w_1 \rightarrow w_1\ R_\alpha\ w)$ does not hold in ANML

4. R_α is transitive iff $\forall w_1, w_2, w_3\ (w_1\ R_\alpha\ w_2\ \&\ w_2\ R_\alpha\ w_3 \rightarrow w_1\ R_\alpha\ w_3)$ does not hold in ANML

5. R_α is euclidean iff $\forall w_1, w_2, w_3\ (w_1\ R_\alpha\ w_2\ \&\ w_1\ R_\alpha\ w_3 \rightarrow w_2\ R_\alpha\ w_3)$ does not hold in ANML

- **Serial**

D. R_α is serial iff $\forall w\ \exists w_1\ (w\ R_\alpha\ w_1)$ does not hold in ANML

$([\alpha]\ A) \rightarrow (<\alpha>\ A)$

The serial property implies that there is always an accessible world and the system does not get into a *false* state. In ANML, an *abort* process is possible leading to a *failed* state. If a process is aborted, there is no accessible world. ANML is therefore not serial.

- **Reflexive**

T. R_α is reflexive iff $\forall w\ (w\ R_\alpha\ w)$ does not hold in ANML

$A \rightarrow [\alpha]\ A$

Counter-example: Let the negotiation state *offered* and the formulae [*agree*]*agreed* and *open* $\leftrightarrow$ one-of({*offered*, *agreed*}) hold in the world w_1. Executing the process

agree leads to an *agreed* and ¬*offered* state in all worlds accessible to w_1. The state *agreed* does not hold in the world w_1 where the state is *offered* and ¬*agreed*, but neither does the state [*agree*] *offered* hold in w_1 even if *offered* holds.

- **Symmetric**

  B. R_α is symmetric iff $\forall w, w_1 (w\ R_\alpha\ w_1 \rightarrow w_1\ R_\alpha\ w)$ does not hold in ANML

  $([\alpha]A \rightarrow [\alpha][\alpha]A)$

Counter-example: Let a protocol include the rules *proposed* → *open*, *agreed* → *closed, proposed* → [*agree*] *agreed*. Let the state *proposed* hold in world w_1. In all worlds w_2 accessible to w_1, the state *agreed* holds from executing an *agree* process in w_1. The protocol allows no possible path from w_2 to w_1 or from w_2 to a world where *proposed* holds i.e. from *agreed* to *proposed*. Symmetry does not hold in ANML, especially with irreversible states such as terminal states. The formula $[\alpha]A \rightarrow [\alpha][\alpha]A$ does not hold since executing a process α leads to another world with a different state where A may not hold and where there may not be a possible path to a world where A holds.

- **Transitivity of R_λ**

  4. R_α is transitive iff $w_1 R_\lambda\ w_2\ \&\ w_2 R_\lambda\ w_3 \rightarrow w_1 R_\lambda\ w_3$ does not hold in ANML

  $([\alpha]A \wedge [\alpha][\alpha]B) \rightarrow [\alpha]B$

Counterexample: If the state in world w_1 is *requested* and λ is the process *offer*. Then let *offered* hold in w_2 from making an *offer* in w_1. In w_2, making an *offer* leads to w_3 where the state is *agreed*. *agreed* is triggered from the state *offered*. However from w_1 and *requested*, R_λ as an *offer* process does not directly lead to w_3 and *agreed*. The relation mappings (*requested* R_{offer} *offered*) & (*offered* R_{offer} *agreed*) does not imply (*requested* R_{offer} *agreed*). From a *requested* state, only executions of the process *offer* twice leads to the *agreed* state. One offer leads to an offered state and two offers lead to an agreement.

- **Euclidean**

5. R_α is euclidean iff $\forall w_1, w_2, w_3 (w_1 R_\alpha w_2 \& w_1 R_\alpha w_3 \rightarrow w_2 R_\alpha w_3)$
   does not hold in ANML
   $([\alpha]A \wedge [\alpha]B \rightarrow [\alpha][\alpha]B)$

Similarly because the relation R_α is not symmetric, ANML is not euclidean where executing process ρ in world w_1 may change the state at all the worlds accessible to w_1.

Counterexample: Let the state *requested* and the formulae *requested*→ <*offer*>*offered* and *requested*→ <*offer*>*rejected* hold in w_1. However this does not imply that from *offered*, an *offer* leads to a *rejected* state or vice versa i.e. *offered*→ <*offer*>*rejected* cannot be inferred.

3.9 Axioms on ANML Connectives

This section gives axioms and inference rules in ANML that are useful for reasoning about a negotiation and planning towards a goal. For each axiom and rule, an example of the application of the rule to a negotiation process is given. Let $\alpha_1 \ldots \alpha_n \in Proc(\phi, \omega, Agents)$, $A \in Fma(\phi, \omega, Agents)$, $Ag_x \in Agent\_group$, $Ag_y \in Agent\_group$, $\varpi \in \omega$. ϖ is an atomic process.

3.9.1 State Axioms

The parameterised sub-state axiom: If state $A(X)$ holds, then the un-parameterised state A also holds. For example, when the state *offered*(X) is *true*, the state *offered* is *true* in that world.

Parameterised sub-state $A(X) \rightarrow A$

Factorisation $[\alpha]A_1 \wedge [\alpha]A_2 \wedge \ldots. \wedge [\alpha]A_n \leftrightarrow [\alpha] (A_1 \wedge A_2 \wedge \ldots. \wedge A_n)$

If a process triggers a conjunction of states, then it triggers each state.
Example of Factorisation axiom: [*X.offer*]*offered*(X) ∧ [*X.offer*]*open* ∧ [*X.offer*]*negotiating* ↔ [*X.offer*](*offered*(X) ∧ *open* ∧ *negotiating*). The process *offer* triggers the state *offered*(X) and its parent states from a protocol.

Disjunctive Factorisation $[\alpha_1]A_1 \vee \ldots. \vee [\alpha_n]A_n \rightarrow [\alpha] (A_1 \vee \ldots. \vee A_n)$

where $\alpha = (\alpha_1 \cup \ldots \cup \alpha_n)$

Disjunctive Factorisation is a weaker axiom that allows non-determinism in a process when any state in a disjunctive formula suffices.

Example of Disjunctive Factorisation: $[X.offer]offered(X) \vee [X.request]requested(X) \vee [X.propose]proposed(X) \rightarrow [X.offer \cup X.request \cup X.propose]$ $(offered(X) \vee requested(X) \vee proposed(X))$.

Intermediate state $[\alpha; \beta]\ state_\beta \leftrightarrow [\alpha];\ state_\alpha?;\ [\beta]\ state_\beta \leftrightarrow [\alpha][\beta]\ state_\beta$

There is an intermediate state that holds between two (or more) processes in a sequence. The intermediate state is the result of executing the first process and is the precondition state for executing the second process. For example, a transaction process consists of the sequence of processes *buy*;*pay* with result *paid*. [*buy*;*pay*]*paid*. The intermediate state is *bought* and is the source state for the *pay* process.

3.9.2 Inference Rules and Axioms over Processes

Let $\alpha_1 \ldots \alpha_n \in Proc(\phi, \omega, Agents)$, $A \in Fma(\phi, \omega, Agents)$, $Ag_x \in Agent\_group$, $Ag_y \in Agent\_group$

Distributive $\alpha; (\beta \cup \delta) \leftrightarrow (\alpha ; \beta) \cup (\alpha ; \delta)$

The sequential operator ";"distributes over the alternation operator "∪". Process alternation does not distribute over ";". Doing a process α following by either process β or δ is equivalent to executing α followed by β or process α followed by δ. For example in an auction, *auctioneer*. *post*;(*auctioneer.withdraw* ∪ *bidder.bid*) is equivalent to (*auctioneer.post*;*auctioneer*. *withdraw*) ∪ (*auctioneer*. *post* ;*bidder.bid*)

Decomposition $$\frac{\alpha_0;\alpha_1;...;\alpha_n = \alpha_0;\delta_1;...;\delta_p}{\alpha_1;...;\alpha_n = \delta_1;...;\delta_p}$$

Example of Decomposition: *browsing*; *choosing*; *negotiating*;*paying* = *browsing*; *shopping*; *delivering* can derive *choosing*; *negotiating*; *paying* = *shopping*; *delivering*

Multiple Alternation $$\frac{\alpha = \alpha_1 \cup \alpha_2 = \alpha_1 \cup \alpha_3}{\alpha = \alpha_1 \cup \alpha_2 \cup \alpha_3}$$

Example of M. Alternation: *shopping* = *auctioning* ∪ *bidding* = *negotiating* ∪ *voting* ∪ *contracting*, then *shopping* = *auctioning* ∪ *bidding* ∪ *negotiating* ∪ *voting* ∪ *contracting*. This rule allows representing all choices in one formula.

Multiple Iteration $$\frac{\alpha_1;\alpha^* = \alpha_1;\lambda^n}{\alpha^* = \lambda^n}$$

Example of M. Iteration: *display*; *browse** = *display*; *request*$^{>10}$, then *browse** = *request*$^{>10}$.

Similar Processes $$\frac{\alpha_1 = \alpha_2, \alpha_1 :: \alpha_3}{\alpha_2 :: \alpha_3}$$

If two processes α_1 and α_2 are similar and given that one process α_1 is subsumed by α_3 then α_2 is constrained by α_3.

For example, the declaration *shopping* = *bidding* where *bidding*::*English-Auction* allows us to infer that the *shopping* process follows an English auction
i.e. *shopping* :: *English-Auction*.

Alternation over instances $(\alpha::\alpha_1 \wedge \beta::\beta_1 \wedge [\alpha \cup \beta]\, A) \leftrightarrow$
$[\alpha]\, A \wedge [\beta]\, A$

The Alternation over instances axiom is the same as the Alternation axiom with the addition that a process instance is constrained. For example, (*shopping*::*bilateral-negotiation* ∧ *auction*::*Dutch-Auction* ∧ [*shopping* ∪ *auction*] *sold*) ↔
[*shopping*]*sold* ∧ [*auction*] *sold*. The two types of processes lead to the same end state.

Sequencing over protocols $(\alpha::\alpha_1 \wedge \beta::\beta_1 \wedge [\alpha;\, \beta]\, A) \leftrightarrow [\alpha]\, [\beta]\, A$

The above axiom concerns the sequencing of processes, typed with another process, towards a state given intermediate states.

Example: Given (*negotiating*::*sealed-bid-auction* ∧ *pay*::*SSL-web* ∧ *posted* ↔ [*negotiating*]*sold* ∧ *sold* ↔[*pay*]*paid*), then [*negotiating*][*pay*]*paid*) and [*negotiating*; *pay*]*paid* can be inferred from the source state *posted*.

Alternation over an agent	$[Ag_x.\alpha \cup Ag_y.\beta]\ A \leftrightarrow\ [Ag_x.\alpha]\ A \wedge [Ag_y.\beta]\ A$
Sequencing over an agent	$[Ag_x.\alpha\ ;\ Ag_y.\beta]\ A \leftrightarrow\ [Ag_x.\alpha]\ [Ag_y.\beta]\ A$

The above two axioms allow to infer alternation and sequencing axioms over processes typed with the agents performing a process.

Process Type $\alpha :: \alpha_1 \wedge [\alpha]\ B \leftrightarrow (\ [\alpha_1]C \vdash [\alpha]B)$

If a process α is constrained by another process α_1, then the states and state transitions allowed in process α can be inferred from the process execution α_1.

State of a tested process $\alpha?\ A \leftrightarrow [\alpha]A$

The state after testing a process is its state after its execution since testing the success of a process involves its execution.

3.10 The state of a process-like negotiation between agents

A negotiation between a group of agents is an abstract process and the state of a negotiation exists only because of the beliefs shared by the agents. The state of an agent is distinguished from what an agent believes to be the state of a negotiation. The state of a negotiation changes depending on the actions of the agents in a group. Changes in the state of a negotiation become part of the individual beliefs of an agent on sending or receiving messages and are eventually propagated to the shared beliefs between the agents as the interaction progresses. The group knows a common protocol and should be aware of the current state of a negotiation and of possible successor actions and states, given by the protocol. On the other hand, the state of an agent differs from the state of a negotiation and remains private to that agent. For example, in a shopping scenario a retailer displays and transacts about goods and may be aware (but not necessarily) of the state of their goods and of own state. A retailer

is not aware of the customers browsing online unless they start interacting. A customer would not know the state at the retailer's side and whether the latter is dealing with other customers. Each agent has its individual and private beliefs useful for strategic reasoning in competitive scenarios.

The state of a negotiation has a hierarchical structure in that a state may contain sub-states. When a state holds, it is necessary that its parent states also holds e.g. when *offered* is true, then *open* is also true if *offered* is a sub-state of *open*. However it is not necessary for a sub-state to be true while its parent state holds. A state conveys information about a negotiation. Partial information is available when it is known that a parent state holds e.g. from knowing the negotiation is *closed*, it is not known in what sub-state of *closed* out of *rejected*, *agreed* or *timedout*. More specific information is obtained by knowing which sub-state is true. A current state means that state which is true while other non-parent states are *false*, as defined by a protocol for the relation between states.

At this point it becomes useful to discuss the concept of a negotiation process and its state. A negotiation process is not as tangible a process as painting or eating. Believing the state of negotiation entails more than just believing a proposition. In modeling the real world and a multi-agent system, a number of propositions can be expressed as holding in that world which are independent of the actions of an agent. For example, the formulae that a duck is a bird, the apple is red or the weather is rainy are not brought about by agents. Even truth formulas such as the door is open, the picture is nailed on the wall, the wall is blue may not be explicitly related to an agent performing a process. These formulae may reflect the state of the world independent of a multi-agent system. In contrast, a negotiation and its state between a group of agents exist because of the agents in that group, their beliefs and actions and reside in the mental states of the participants. A negotiation does not exist before or after the group of agents become involved in it. The states and process of negotiation depends on the existence of the involved group. A negotiation process and its state are as abstract notions.

The protocol of a negotiation is assumed to be commonly known by all the participating agents. Joint (implicit) commitment by these agents to a protocol entails

complying to that protocol while executing a negotiation process. Non-compliant agents are ignored but compliance benefits an agent or a group of agents for meaningful interaction towards achieving a goal. A negotiation protocol is represented as a logical theory in ANML with relation between states and with condition-action rules. A protocol may be commonly known by a group of agents through for example learning about it, a public repository or via another negotiation. The state or status of a negotiation is considered to be the formulae that are rendered true by message exchange between involved agents. A complete protocol in ANML specifies the initial status of a negotiation - that is, what are the truth values of the formulae with respect to a negotiation process before its start. A protocol also specifies which new state of the negotiation is triggered after an agent sends a message. In a complete protocol, transitions would render some formulae *true* and others *false*.

Therefore, a negotiation process can be considered as a sequence of messages being exchanged because a group of agents has agreed to follow a protocol. A negotiation is a temporary process that exists only as long as the agents are committed to that protocol. A negotiation state exists only because the participants all have beliefs about the propositions (concerning that negotiation) that hold in the world at that instance. We say that the state of a negotiation is *S*, if all the participating agents believe the formula *S* to hold in that world. For example the state of a negotiation is *offered* if all the participants believe that proposition *offered* holds in the world. An agent is capable of introspection. The truth-values of the formulae and propositions are determined by a process and the protocol. There is the question of relating common beliefs to the negotiation state, for example an agent believing other agents believe the state and so forth. (Chapter 7 further discusses agents modeling other participants' beliefs).

Before a negotiation, in world w_0, involved agents believe the initial state, s_0, of a negotiation process, *p*. When an initiator sends the first message, *m*, there is a new set of formulae, s_1, that holds because *m* leads to the world w_1. When beliefs about s_1 being true become shared by all the agents, then the state of the negotiation has changed from s_0 to s_1. s_0 and s_1 are the states holding in the worlds w_0 and w_1 respectively. During a negotiation, the state of the negotiation changes according to

the messages being exchanged. On receiving a message, n, in world w_n, an agent X changes its belief about the set of propositions holding in the new world, w_{n+1}, according to the protocol being followed. The new set of propositions entails that an agent believes the negotiation state to be s_{n+1}. All agents eventually share the same beliefs that the negotiation state is s_{n+1} in world w_{n+1}. Beliefs about the negotiation state change after each process or message and processes enable state transitions concerning a negotiation instance. Finally a terminal action in a protocol is regarded as being the final process. A negotiation ends in a terminal state when all agents have the same beliefs about the last formulae holding.

A negotiation process is thus abstract and temporary since once the agents know that the terminal state of a negotiation holds, the negotiation process resides in their history. The negotiation, as a series of messages being exchanged, has only meaning within that group. The result of a negotiation may give rise to consequences and contracts. No agent has a view of the whole negotiation process since each agent has its own private beliefs about the negotiation. Only an omniscient agent having an overall view of the negotiation is able to infer when the current state of negotiation comes to be wholly shared.

An agent's modeling of other agents depends on the history and prediction of their interaction according to the protocol. An agent partially believes how a process may affect its opponents' beliefs and behaviour. A rational agent models the environment and the behaviour of other agents so as to minimise its uncertainty about reaching its goal. Uncertainty because of fallible communication and of recent events and states fade with progress of a negotiation. (See chapter 7 for further discussion on reasoning about the consistency between the beliefs of agents in a group given uncertainties and failures in the underlying communication medium.)

3.11 Further Work on ANML

The meta-language ANML is suitable for expressing agent negotiations and protocols for them, since it is an intuitively computational form. More work can be carried out on ANML through automated reasoning and specifying the semantics of protocols. The meta-language ANML may inherit from the properties of PDL such as soundness

and completeness. The completeness proof for PDL inherits from the Quasi-Henkin completeness proof for modal logics, [Ben-Ari et al 1982]. See Goldblatt [1992] for the properties of PDL. Further work includes analysing how the properties of PDL reflect into ANML.

ANML allows non-deterministic choice as it permits more than one choice of next move at some step in a computation. For example in a bilateral negotiation there are several possible transitions from a *requested* state. ANML allows $s_1 \rightarrow [\alpha \cup \beta]\ S_2$ where either process α or β may be performed from state s_1.

3.12 Conclusion

This chapter surveys various logics for specifying negotiation processes and protocols. A multi-modal language such as propositional dynamic logic is extended to specify a meta-language called ANML. This chapter gives the syntax, semantics, inference rules and axioms in ANML. Like PDL, ANML is a normal modal system. Such a language can be applied to represent an abstract process like negotiation among a group of agents following a protocol. The current state of a negotiation resides in the shared beliefs of the group.

The meta-language ANML is an application of theoretical work on logic languages to expressing a negotiation between agents. The rest of this thesis describes how to apply ANML for representing, verifying and reasoning about protocols for negotiation, with testable properties. The next chapter provides several examples of expressing various negotiation protocols in ANML.

4 Representing Protocols in ANML

4.1 Introduction

An interaction protocol specifies a pattern of message exchange that two or more agents may follow in communicating with one another. It may be regarded as the set of public rules or guidelines indicating the conduct of an agent towards other agents when carrying out some negotiation. A commonly known protocol ensures that all participants following it will coordinate meaningfully and can expect certain responses from others. A protocol can be application or domain specific, but as long as all the participants know and follow it, a conversation can be carried out, which may eventually end up in a terminal state. We adopt the approach of a protocol being commonly known to all participants. Possible scenarios where a group of agents may learn a protocol include:

- A public repository of standardised protocols and their semantics where agents can decide which protocol to use and share.
- One or more of the agents advertise that they are willing to negotiate using a particular protocol.
- The negotiation is performed in the framework of an electronic institution, which publishes the protocols it supports.
- There is a pre-negotiation phase between a group of agents to agree on the protocol to follow for the following negotiation about a service.

The previous chapter specified a normal modal system for ANML as an extension of propositional dynamic logic. This chapter shows the application of the meta-language to represent interaction protocols, [Paurobally and Cunningham 2000a]. Protocols are

expressed in statecharts and ANML. Statecharts provide an initial illustration of the protocols. ANML is used for a complete and unambiguous specification of the protocol.

Statecharts, [Harel and Namad 1996], are a graphical method to illustrate reactive behaviour over time and are an extension of conventional finite-state machines and state transition diagrams. The statecharts presented in this chapter show the relationships between the states of a negotiation process and permitted transitions between the states, so implicitly define a negotiation protocol (which is made explicit in its ANML logical theory). These protocols can form the basis for further customisation for application or domain specific interactions, however, even after augmenting a statechart to type an action with the agent executing it, it is shown that the statechart of a realistic protocol is often not complete and is prone to a number of errors. Statecharts typed with agents do not correctly capture features such as synchronisation between two agents, parameterisation of processes with agents and iterative processes.

Studied protocols are explained through multi-agent scenarios showing the progress of a corresponding negotiation. As described in the previous chapter, we refer to *state* is short for the *state of a negotiation* by virtue of being believed by all agents in a group and does not mean the state of individual agents. The group also knows the protocol they all mean to comply with. Agents in a group follow a protocol to negotiate on a subject of negotiation such as a service. The negotiation subject may consist of an aggregation of various issues. The knowledge of a group of agents includes a protocol and the beliefs include the current state of a negotiation and a negotiation subject.

In practice, an agent may be involved in several concurrent negotiations and often needs authorisation to become a member of a group or join in any negotiation. Each negotiation process is associated with its participants, a subject of negotiation, a state and a protocol. A negotiation subject may be a tuple or list of issues and is passed as a parameter in exchanged messages. A message may be in the form of a communicative act including a performative or process such as *offer* and a subject content (such as car, £2000, etc.). An agent is able to distinguish between multiple

negotiations so that it can relate a process to its state and choose an appropriate action. Implicitly each negotiation state is characterised with a tuple (*negotiation instance*, *participants*, *protocol*, *subject*) and each negotiation action is implicitly associated with another tuple (*negotiation instance*, *perpetrator of that action*, *protocol*, *subject*).

4.2 A Bilateral Protocol

A bilateral protocol defines a protocol of conversation between two parties looking for an agreement over a negotiation subject. The initial version of the bilateral protocol studied here is expressed as a state transition diagram in [OSM SARL 1998]. This section describes the corrected version of this bilateral protocol, (the original protocol is presented in chapter 5), and illustrates three scenarios where two agents negotiate by complying to that protocol. Its statechart is given in Figure 4.1 and its ANML is codified as Theory 4.1.

According to Figure 4.1, entry in a bilateral negotiation is through either an *initial-request*, *initial-offer* or *initial-propose* message leading to an *open* state and more precisely to a *requested*, *offered* or *proposed* state respectively. The process *X.initial_request* means that agent *X* sent an *initial_request* leading to a *requested*(*X*) state. The state *requested*(*X*) is read as agent *X* has triggered the state *requested*. Each state may be interpreted as conveying a level of commitment towards an agreement. For example an *agreed* state entails more commitment than a *requested* state. A *reject* action or *timeout* event can occur at any time. From a *requested* state, both agents can continuously make *suggest* actions to remain in that state while modifying the subject of negotiation until one of them wants to move to a higher level of commitment through an *offer* or *propose*. The *proposed* state is a sub-state of *offered* and both of these states allow an agreement to follow in the next action leading to an *agreed* state. The difference between *offered* and *proposed* is that from the former state, an agent can only *agree* or *reject* whereas from the *proposed* state, an agent may *agree*, *reject* or return to a *requested* state through a *request*.

Figure 4.1 does not show full parameterisation of states since it does not reflect the fact that agents *X* or *Y* making a *suggest* depends on which agent triggered the last

state. It is hard to express parameterisation of states and actions in a statechart when there are both non-iterative and iterative processes leading to the same state.

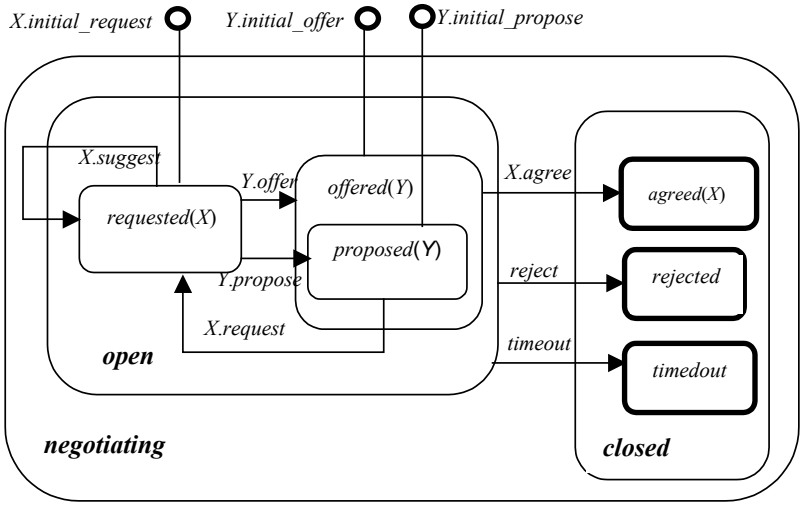


Figure 4.1 State transition diagram for bilateral protocol

$\neg negotiating \leftrightarrow [\{X,Y\}.bilateral\_process_b]\ closed$ (1)

$negotiating \leftrightarrow$ one-of ({ *open*, *closed* }) (2)

$open \leftrightarrow$ one-of ({ *requested*, *offered* }) (3)

$closed \leftrightarrow$ one-of ({ *agreed*, *rejected*, *timeout* }) (4)

$proposed(X) \rightarrow offered(X)$ (5)

$\neg negotiating \leftrightarrow$ none-of ({ *open, closed* }) (6)

$\neg open \leftrightarrow$ none-of ({ *requested*, *offered* }) (7)

$\neg closed \leftrightarrow$ none-of ({ *agreed*, *rejected, timeout* }) (8)

$\neg negotiating \leftrightarrow [X.initial\_request] requested(X) \vee [X.initial\_offer] (offered(X) \wedge \neg proposed(X)) \vee [X.initial\_propose]\ proposed(X)$ (9)

$requested(X) \leftrightarrow [Y.offer]\ (offered(Y) \wedge \neg proposed(X)) \vee [Y.propose]\ proposed(Y) \vee [Y.suggest]\ requested(Y) \wedge \neg(X = Y).$ (10)

$offered(X) \leftrightarrow [Y.agree]\ agreed(Y) \wedge \neg(X = Y).$ (11)

$proposed(X)) \leftrightarrow [Y.request]\ requested(Y) \wedge \neg(X=Y).$ (12)

$open \leftrightarrow (\ [X.reject]\ rejected \vee [timeout]timedout \vee [offered(X)]?;\ Y.agree]\ agreed(Y)\ ) \wedge \neg(X=Y).$ (13)

Theory 4.1 Bilateral Negotiation Protocol in ANML

Rule (1) defines process *bilateral_process*$_b$ as an instance of a joint negotiation between agents *X* and *Y*, following a bilateral protocol with initial state ¬*negotiating*. After executing the process *bilateral_process*$_b$, the state is *closed*.
Axioms (2) to (8) give the relation between state and sub-states. The overall parent state is *negotiating*. The states *open* and *closed* are sub-states of *negotiating*. A negotiation can be either *open* or *closed* but not in both states simultaneously, making the two sub-states mutually exclusive. The double implication in axiom (2) ensures that the *negotiating* parent state does not hold without being in one of its sub-states. Likewise, *open* and *closed* themselves consist of mutually exclusive sub-states. If the state is *open* then either *requested* or *offered*, but not both, holds and vice versa. A single implication in axiom (5) implies that *proposed* is a sub-state of *offered*, but *offered* does not depend on its sub-state. A negotiation state can be *offered* without being *proposed*.

Axioms (2)-(4) ensure that only the current state triggered by a process is *true* while other non-parent states are *false*. Axioms (6)-(8) do not allow any sub-states to hold when their parent states do not hold in a world. When a parent state does not hold, axiom (6)-(8) ensure that it is not because two or more of its sub-states hold.

Axiom (9) specifies the three entry actions when a negotiation has not started i.e. when the overall parent state, *negotiating*, does not hold. Axioms (10) to (13) are action-condition rules representing the effect of processes on states. Contrary to the statechart in Figure 4.1, the ANML protocol fully parameterises the processes and the states with the agents performing a process. Axiom (13) defines the possible actions to terminate a negotiation from an *open* to a *closed* state. To do so, either agent can reject, a timeout can occur or an agent agrees to the other agent's offer.

4.2.1 Buying a pizza online (Business to Consumer)

The first scenario for explaining the bilateral protocol involves *John* buying a pizza online from *Lorenzo*'s Pizzeria. The latter has a web page where any buyer can browse through a catalogue of Italian food and delivery services. *Lorenzo*'s Pizzeria has an agent, called *lorenzo* that acts as a seller and *John*'s agent, called *john*, acts as a buyer on his behalf. Both agents join in a negotiation according to the above bilateral protocol with ¬*negotiating* as initial state. The catalogue lists that *Lorenzo*'s Pizzeria

offers a medium pizza with two toppings and free delivery for £7.00. The issues in the negotiation subject are (*size*, *name of product*, *number of toppings*, *price*, *delivery*) and the negotiation is started with an *initial-offer* from *lorenzo*. From axiom (9) in Theory 4.1, the process *lorenzo.initial-offer* triggers a transition of the negotiation state from ¬*negotiating* to (*offered*(*lorenzo*) ∧ ¬*proposed*(*lorenzo*)). According to the bilateral protocol, the following two action-condition rules can fire from a (*offered*(*lorenzo*) ∧ ¬*proposed*(*lorenzo*)) state.

offered(*lorenzo*) ↔ [*john.agree*] *agreed*(*john*) ∧ ¬(*john* = *lorenzo*).
open ↔ [*john.reject* ∪ *lorenzo.reject*] *rejected*, [*timeout*]*timedout*,
[*offered*(*lorenzo*))?; *john.agree*] *agreed*(*john*) ∧ ¬(*lorenzo*=*john*).

John can only agree to or reject *lorenzo*'s offer or the negotiation may timeout. In this scenario, the negotiation terminates successfully with *john* agreeing to *lorenzo*'s offer. The action *john.agree* changes the state from *offered*(*lorenzo*) to *agreed*(*john*). Scenario 4.1 shows the interaction between *lorenzo* and *john* and changes in the negotiation sub-state after each message.

Step number	**Agent performing action**	**Action**	**Resulting negotiation state**
			¬*negotiating*
1	Agent *lorenzo* from *Lorenzo*'s Pizzeria	*initial_offer* medium, pizza, two toppings, £7:00, free delivery	*offered*(*lorenzo*) medium, pizza, two toppings, £7:00, free delivery
2	Agent *john* for client *John*	*john.agree* medium pizza, two toppings, £7:00, free delivery	*agreed*(*john*) Agreement: Lorenzo delivers a medium pizza, two toppings for £7:00 to *John*

Scenario 4.1 *John* buying a pizza from *Lorenzo*'s Pizzeria

4.2.2 Buying telecommunication bandwidth (Business to Business)

The second scenario describes trading bandwidth against insurance services between two companies, *Imperial* telecom and *Leicester* Co. *Imperial* telecom provides bandwidth for different types and number of connection. It can also provide portal and mobile facilities. *Leicester* Co. provides insurance for a number of employees with an upper limit on their salaries. The subject of negotiation consists of the amount of bandwidth and insurance to be exchanged according to the bilateral protocol. The interaction starts with the agent from *Leicester* Co., *leicester*, making an *initial_request* for bandwidth covering 40 PCs against a group insurance for 15 people. After several proposals, suggestions and requests, *leicester* makes an offer to provide insurance for 20 people at 70% of their salary with a maximum of £2500 per month against obtaining bandwidth for 40 ISDN connections at 64-128 kps and portal and mobile facilities from *Imperial* telecom. The latter terminates the negotiation with a rejection. This scenario differs from the usual consumer provider interaction in that it involves two businesses exchanging non-tangible services where price is not an issue.

Agent doing action	Action	Resulting negotiation state
Leicester Co.	*leicester.initial_request* bandwidth for business, 40 PC against group insurance for 15 people.	*requested*(*leicester*) bandwidth for business, 40 PC against group insurance for 15 people
Imperial telecom	*imperial.propose* 40 modem connections <56kps or 30 ISDN <128 kps or 20 connections for ADSL lines <8 Mbps against group insurance for 15 people	*proposed*(*imperial*) 40 connections for Modem <56kps or 30 ISDN <128 kps or 20 connections for ADSL lines <8 Mbps against group insurance for 15 people
Leicester Co.	*leicester.request* 40 ISDN connections at 64-128 kps against group insurance for 20 people	*requested*(*leicester*) 40 ISDN connections at 64-128 kps against group insurance for 20 people

Imperial telecom	*imperial.propose* 40 ISDN 64-128 kps against group insurance for 20 people with disability benefits	*proposed*(*imperial*) 40 ISDN 64-128 kps against group insurance for 20 people with disability benefits
Leicester Co.	*leicester.request* 40 ISDN connections at 64-128 kps against group insurance for 20 people with first day hospitalisation coverage.	*requested*(*leicester*) 40 ISDN connections at 64-128 kps against group insurance for 20 people with first day hospitalisation coverage.
Imperial telecom	*imperial.suggest* 40 ASDL connections at 0.5-8 Mbps against group insurance for 40 people with first day hospitalisation coverage	*requested*(*imperial*) 40 ASDL connections at 0.5-8 Mbps against group insurance for 40 people with first day hospitalisation coverage
Leicester Co.	*leicester.suggest* 40 ASDL 0.5-8 Mbps against group insurance for 25 people, disability benefits, 70% of salary	*requested*(*leicester*) 40 ASDL 0.5-8 Mbps against group insurance for 25 people, disability benefits, 70% of salary
Imperial telecom	*imperial.suggest* 30 ASDL 0.5-8 Mbps against group insurance for 25 people, disability benefits, 70% of salary	*requested*(*imperial*) 30 ASDL 0.5-8 Mbps against group insurance for 25 people, disability benefits, 70% of salary
Leicester Co.	*leicester.propose* 40 ISDN 64-128 kps against group insurance for 20 people, 59% of salary and employee compensation wages	*proposed*(*leicester*) 40 ISDN 64-128 kps against group insurance for 20 people, 59% of salary and employee compensation wages
Imperial telecom	*imperial.request* 40 ISDN 64-128 kps + mobile facilities against group insurance for 20 people, 70% of salary and employee compensation wages	*requested*(*imperial*) 40 ISDN 64-128 kps + mobile facilities against group insurance for 20 people, 70% of salary and employee compensation wages
Leicester Co.	*leicester.propose*	*proposed*(*leicester*)

	40 ISDN 64-128 + mobile facilities against group insurance for 20 people, 70% of salary and maximum salary £2000 per month	40 ISDN 64-128 + mobile facilities against group insurance for 20 people, 70% of salary and maximum salary £2000 per month
Imperial telecom	*Imperial.request* 40 ISDN 64-128 + mobile facilities +portal against group insurance for 20 people, 70% of salary and maximum salary £3000 per month	*requested*(*imperial*) 40 ISDN 64-128 + mobile facilities +portal against group insurance for 20 people, 70% of salary and maximum salary £3000 per month
Leicester Co.	*Leicester.offer* 40 ISDN 64-128 + mobile facilities +portal against group insurance for 20 people, 70% of salary and maximum salary £2500 per month	*offered*(*leicester*) 40 ISDN 64-128 + mobile facilities +portal against group insurance for 20 people, 70% of salary and maximum salary £2500 per month
Imperial telecom	*Imperial.reject* 40 ISDN 64-128 + mobile facilities +portal against group insurance for 20 people, 70% of salary and maximum salary £2500 per month	*rejected* 40 ISDN 64-128 + mobile facilities +portal against group insurance for 20 people, 70% of salary and maximum salary £2500 per month

Scenario 4.2 Exchange of bandwidth and insurance between 2 companies

4.2.3 Holiday (Business to Consumer)

Oliver wants to buy a package holiday to Mauritius from the travel agent *Beach* Hols. Their two agents, *oliver* and *beach*, negotiate on the holiday package according to a bilateral protocol and *Oliver* aims to fly to Mauritius in August for a fortnight. The subject of negotiation is a holiday package to Mauritius for a certain number of days during peak season. *Oliver* starts the interaction with an *initial_request* for a holiday to Mauritius for 2 weeks. The negotiation ends with an agreement between *Oliver* and the travel agent for a holiday to Mauritius between 10th to 24th August, flights and hotel accommodation included at £1600.

Agent performing action	Action	Resulting Negotiation state
Oliver	*oliver.initial_request* Package holiday to Mauritius for 2 weeks	*initial_request*(*oliver*) Package holiday to Mauritius for 2 weeks
Beach Hols	*beach.propose* *H* = holiday to Mauritius, 14 days, July, flights + hotel, £1200	*proposed*(*beach*) *H*
Oliver	*oliver.request* *H1*= holiday to Mauritius, August, 14 days, flights + hotel, £1200	*requested*(*oliver*) *H1*
Beach Hols	*beach.suggest* *H2* = holiday to Mauritius, August, 14 days, flights + 3* hotel, £2000	*requested*(*beach*) *H2*
Oliver	*oliver.suggest* *H3* = holiday to Mauritius, 10/08- 24/08, flights + hotel <£1500	*requested*(*oliver*) *H3*
Beach Hols	*beach.suggest* *H4* = holiday to Mauritius, 10/08- 24/08, flights + 2* hotel +car hire, £1800	*requested*(*beach*) *H4*
Oliver	*oliver.propose* *H5* = holiday to Mauritius, 10/08- 24/08, flights + 2* hotel , £1600	*proposed*(*oliver*) *H5*
Beach Hols	*beach.agree* holiday to Mauritius, 10/08- 24/08, flights + 2* hotel, £1600	*agreed*(*beach*) *Beach* Hols sells holiday to Mauritius, 10/08- 24/08, flights + 2* hotel, £1600 to *Oliver*

Scenario 4.3 *Oliver* and *Beach* Hols negotiating for a package holiday

4.3 Bilateral Negotiation Expanded

In current electronic commerce scenarios, the interaction between a consumer and a business is usually straightforward with phases for information collection, agreement and execution of service in a transaction. A consumer finds it easier to trust an agent to shop online for him when any possible loss incurred will be small. When the risk is large or when a substantial amount of money or goods are involved in a transaction, the owners would prefer their agent to be semi-autonomous and to report to them before any commitment. Trust and acceptance of and between autonomous agents are issues needing to be resolved for large-scale agent deployment. Delegating tasks to agents are more likely to occur first in business to business transactions, where complex and parallel trading are executed over long period of times or in the space of seconds. Business agents should be capable of coping with parallel decision making in automated negotiation and brokerage. Scenario 4.2 showed a business to business negotiation between a telecom and an insurance company. Such a bilateral negotiation, Theory 4.1, can be expanded for richer interactions.

For example consider a simple business to business negotiation between two agents belonging to a web page designer and to Sam Company, which wants a web site. Sam Company's agent, called Sam, has initially consulted a broker agent to get a list of suitable web designers. Sam chooses a web designer called Cool Pages for further transaction. Sam may either contact the agent from Cool Pages directly or use the broker agent as intermediary. Here, direct interaction is considered between Sam and the web designer's agent called Pages. Agent Pages offers a number of features in designing web pages like clickable maps, multimedia plug-ins, secure online transactions, etc. in addition to after-sales maintenance and updates. Agent Sam contacts agent Pages with vague requirements for a web site and agent Pages suggests various designs that might interest his customer. The two agents negotiate on a number of issues such as price, quality of service, number of web pages and after-sales facilities. The agents may follow the bilateral protocol in Theory 4.1, but it may prove to be too constrained and not offer enquiry and argumentation capabilities. Since agent Sam is not an expert in web designing, he may not understand a proposal

or the worth associated to the features on a web page. Sam can ask agent Pages for advice and explanation. Both agents can interact in a dialogue style in order to augment their beliefs about the domain and each other and to reach a mutually suitable solution. Sam gradually refines its requirements for a web page given its needs and Agent Pages can advice and persuade Sam to what would be a solution beneficial to both of them. An agreement may be sealed with a contract, in which the policies for secure payment and delivery are drafted. Since the web pages need to be maintained and updated in the future, agent Pages offers after-sales services which may be part of the initial package or need ad-hoc negotiations. This scenario shows a case when a protocol for a negotiation must allow for richer dialogues and explanations. A JSD diagram [Jackson 1975], is first given and then a state-chart for a protocol that supports richer interactions than Theory 4.1. These two diagrams are easier to understand at first glance, followed by a corresponding logical theory.

4.3.1 Expanded Bilateral protocol in JSD

The JSD diagram in Figure 4.2 shows a sequence of tasks in a business to business transaction as that described in the above scenario. An online transaction consists of an agent, *AgentX*, finding a partner through a broker. Then *AgentX* and its partner engages in a joint task, *make transaction*.

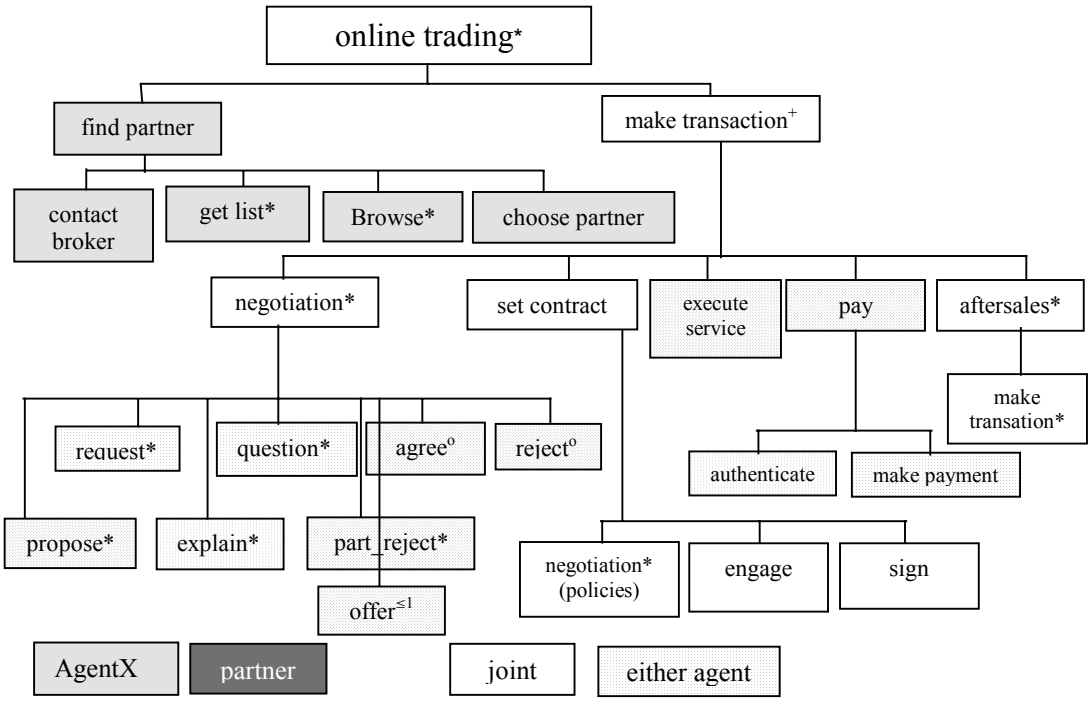


Figure 4.2 JSD of business to business transaction

AgentX obtains a list of partners (providers or customers) through its broker. It browses through the list, chooses a suitable partner and enters in a transaction one or more times with its partner. The joint task *make-transaction* consists of a sequence of tasks – joint negotiation, joint contract, executing a service, payment and joint after-sales activities. A joint negotiation may follow an expanded bilateral protocol allowing questions and explanations. If a sub-task fails, the participants can restart the parent task or the overall transaction. Figure 4.2 does not explicitly represent an interaction protocol but the paths of process execution in a trading scenario. The sequence of tasks in Figure 4.2 is expressed as a path in ANML.

online_trading = (*AgentX.find_partner*?;{*AgentX*, *Partner*}*.make_transaction*$^{+}$)*

*AgentX***.** *find_partner* = *AgentX.contact_broker* ; *AgentX.get_list**;

*AgentX.browse**; *AgentX.choose_partner*

After finding a partner, AgentX and its partner enters in a transaction:

{*AgentX*, *Partner*}*.make_transaction* =

{ *AgentX*, *Partner* }.(*negotiation** ; *agreed*?; *set_contract*; *execute_service* ; *pay*;

*after_sales**

{ *AgentX*, *Partner*}*.negotiation* =

(*AgentX*.(*request* ∪ *propose* ∪ *question* ∪ *explain* ∪ *part_reject*); *Partner*.(*request* ∪ *propose* ∪ *question* ∪ *explain* ∪ *part_reject*)**)*** ; **((***AgentX*. *offer*; one-of({*Partner.reject*, *Partner.accept*})) ∪ (*Partner. offer*; one-of({*AgentX.reject*, *AgentX.accept*}))**)**

When negotiating, either agent can send requests, proposals, ask questions and give explanations. An agent is able to partly reject the issues in a proposal or request and its opponent then knows not to suggest or *propose* these issues. Such dialogues may lead a group to concessions that are beneficial to all agents or bring about a more optimal solution than when using a restricted protocol. After iterations of requests, proposals, questions and explanations, the agents converge to a vector of mutually satisfying values. The negotiation is terminated with a total rejection or an agreement.

{*AgentX*, *Partner* }.*set_contract* = { *AgentX*, *Partner* } .((*negotiation*?)*;
engage; sign)

If both agents have reached an agreement, a contract is drafted. They negotiate on the policies to support the contract such as payment method, security, delivery, notary services and signatures. Bilateral protocols may be used for agreeing upon these policies, after which the two agents jointly engage into and sign the contract.

c.pay = *c.authenticate*; *c.make_payment*

Both agents can execute the terms of a contract through payment and delivery with suitable security mechanisms like identification, authentication and non-repudiation steps.

{*AgentX* , *Partner*}.*after_sales* = {*AgentX* , *Partner*}.*make_transaction*

If the contract does not already contain terms for after-sales services, both agents may start a transaction about maintenance or updates in the future, similar to the *make_transaction* process. The next section first gives an expanded bilateral protocol as a state-chart then in ANML.

4.3.2 State-chart and ANML theory for expanded bilateral protocol

Figure 4.3 illustrates expanding the bilateral protocol in Theory 4.1 for richer interactions than *request, propose*, *suggest*, *offer*, *agree* and *reject* messages.

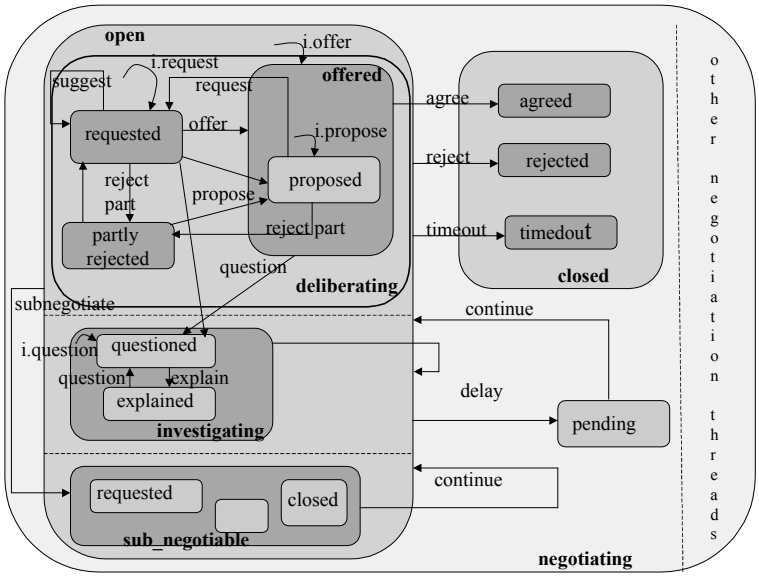


Figure 4.3 Richer bilateral protocol

The *open* state of the bilateral protocol in Figure 4.1 resembles the *deliberating* state in Figure 4.3. Figure 4.3 includes *deliberating*, *investigating* and *sub-negotiating* as

sub-states of *open*. In the *investigating* state, each agent may ask questions and give explanations. The *investigating* state co-exists with the *deliberating* and *sub_negotiating* states e.g. both *requested* and *questioned* may hold. Exiting from an *investigating* state resumes a negotiation into an *open* state. An agent can partly-reject issues it dislikes but can still continue negotiating, with both parties keeping in mind not to use the rejected issues.

The states *sub_negotiating*, *other negotiation threads* and *pending* support nesting of protocols, concurrency and delays respectively. An agent may fork into sub-negotiations, whose states can influence the parent negotiation. This allows the nesting of protocols and the ability to abstract from a parent process into sub-processes for dealing with sub-goals. A negotiation process may be suspended by triggering the *pending* state and later to be continued in its latest *open* state. Concurrent processes are possible between two agents, where they are involved in several parallel negotiations with each other. Both agents can use the proceedings of other negotiations to influence their decisions. Figure 4.3 does not show all the earlier discussed features in an expanded protocol e.g. returning to latest states after *pending* or *investigating*. All the possible actions in the logical theory of the expanded bilateral protocol are captured below.

The overall parent state is *negotiating* consisting of *open* or *closed*. An *open* state can be in *deliberating*, *investigating*, *pending* or *sub_negotiating* sub-states.

$negotiating \leftrightarrow$ one-of($\{open,\ closed\}$)

$closed \leftrightarrow$ one-of($\{agreed\ ,\ rejected\ ,\ timedout\}$)

$open \leftrightarrow (deliberating \vee investigating \vee pending \vee sub\_negotiating)$

$deliberating \leftrightarrow$ one-of($\{\ requested, partly\_rejected, offered\}$)

$investigating \leftrightarrow$ one-of($\{\ questioned, explained\}$)

$proposed(Y) \rightarrow offered(Y)$

$on\text{-}hold \leftrightarrow pending \vee investigating \vee sub\_negotiating$

$\neg\ negotiating \leftrightarrow$ none-of($\{open,\ closed\}$)

$\neg\ closed \leftrightarrow$ none-of($\{agreed, rejected, timedout\}$)

$\neg\ deliberating \leftrightarrow$ none-of($\{\ requested\ , partly\_rejected, offered\}$)

$\neg$ *investigating* $\leftrightarrow$ none-of({ *questioned*, *explained*})

As for the bilateral protocol, the *negotiating* state is entered by an *initial_request, initial_offer* or *initial_propose* and terminates in a *closed* state. An agent may also start a negotiation with an *initial_question*.

$\neg$*negotiating* $\leftrightarrow$ [{*X*,*Y*}.*expanded_bilateral_process*$_b$] *closed*

$\neg$*negotiating* $\leftrightarrow$ [*X.initial_request*]*requested*(*X*) $\vee$ [*X.initial_offer*] (*offered*(*X*) $\wedge$ $\neg$*proposed*(*X*)) $\vee$ [*X.initial_propose*] *proposed*(*X*) $\vee$ [*Y.initial_question*] *questioned*(*Y*)

A question may be asked from both *requested* and *offered* states or issues might be rejected from *requested* or *proposed* states. An *open* negotiation can be suspended in the *pending* state.

(*requested*(*X*) $\wedge$ $\neg$*on-hold*) $\leftrightarrow$ [*Y.offer*] (*offered*(*Y*) $\wedge$ $\neg$*proposed*(*X*)) $\vee$ [*Y.propose*] *proposed*(*Y*) $\vee$ [*Y.suggest*] *requested*(*Y*) $\vee$ [*Y.reject_part*] *partly_rejected*(*Y*) $\vee$ [*Y.question*] (*questioned*(*Y*) $\wedge$ *requested*(*Y*)) $\wedge$ $\neg$(*X*= *Y*).

(*offered*(*X*) $\wedge$ $\neg$*on-hold*) $\leftrightarrow$ [*Y.agree*] *agreed*(*Y*) $\vee$ [*Y.question*] (*questioned*(*Y*) $\wedge$ *offered*(*Y*)) $\wedge$ $\neg$(*X*= *Y*).

(*proposed*(*X*) $\wedge$ $\neg$*on-hold*) $\leftrightarrow$ [*Y.request*] *requested*(*Y*) $\vee$ [*Y.reject_part*] *partly_rejected*(*Y*) $\vee$ [*Y.question*] (*questioned*(*Y*) $\wedge$ *proposed*(*Y*)) $\wedge$ $\neg$(*X*=*Y*).

open $\leftrightarrow$ **(**[*X.reject*] *rejected* $\vee$ [*timeout*]*timedout* $\vee$ [(*offered*(*X*) $\wedge$ $\neg$*on-hold*)?; *Y.agree*] a*greed*(*Y*) **)** $\vee$ [*StateX* ?; (*X.delay* $\cup$*Y.delay*)] (*pending* $\wedge$ *StateX*) $\vee$ [*StateX*?; {*X*, *Y*}.*subnegotiate*] (*sub_negotiating* $\wedge$ *StateX*) $\wedge$ $\neg$(*X*=*Y*).

A parent negotiation may continue after performing a sub-negotiation, returning back to its previous state before entering the sub-process.

$(sub\_negotiating \wedge StateX) \leftrightarrow [X.continue \cup Y.continue]\ (StateX \wedge \neg sub\_negotiating)$

Any agent may continue a negotiation from the *suspended* or *partly_rejected* state.

$partly\_rejected(X) \leftrightarrow [Y.propose]\ proposed(Y) \vee [Y.request]\ requested(Y)$

$(pending \wedge StateX) \leftrightarrow [X.continue \cup Y.continue]\ (StateX \wedge \neg pending)$

In the *investigating* state, questions and explanations may be successively exchanged between agents. From the *investigating* state, a *continue* action resumes the negotiation to the latest sub-state of *open*, which is either *requested* or *proposed*.

$questioned(X) \leftrightarrow [Y.explain]explained(Y) \wedge \neg(X=Y).$

$explained(X) \leftrightarrow [Y.question]questioned(Y) \wedge \neg(X=Y).$

$(investigating \wedge requested(X)) \leftrightarrow [X.continue \cup Y.continue]\ (requested(X) \wedge \neg investigating)$

$(investigating \wedge proposed(X)) \leftrightarrow [X.continue \cup Y.continue]\ (proposed(X) \wedge \neg investigating)$

4.4 Multi-lateral Protocol

The OMG EC digital architecture specification [OSM Saarl 1998] defines state transition diagrams for multi-lateral (Figure 4.4) and promissory negotiation (Figure 4.7).

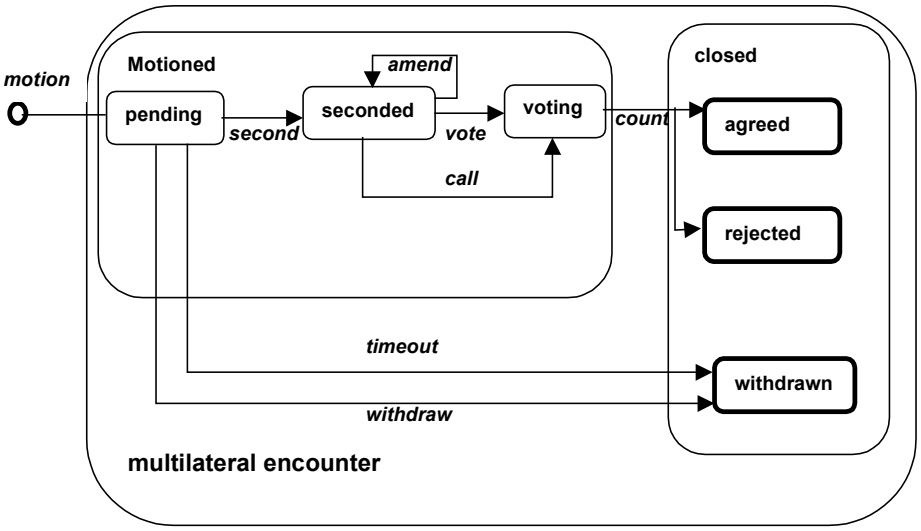


Figure 4.4 Original version of multi-lateral protocol, [OSM Saarl 1998]

The multilateral transition diagrams in Figure 4.4 and Figure 4.5 specifies a protocol relevant for submitting motions in a quorum, for seconding and amending these motions, and for subsequent voting within a community of two or more agents. An agent initiates the multilateral process into a *pending* state by raising a motion on a subject. The initiator can withdraw its motion or the motion may time out leading to a *withdrawn* state. Otherwise from a *pending* state, a *seconded* state is triggered by another participant seconding the motion. In the *seconded* state, a countdown to a vote timeout is activated. Any user may invoke the amend transition in the *seconded* state to change the subject of negotiation or may call a transition to the *voting* state. If the *amend* transition fails, then the state remains *seconded* without any change in subject of negotiation, otherwise if the *amend* succeeds, then the *seconded* state is entered again with a new subject of negotiation and the countdown to voting reinitialised. If the *call* to voting fails, the current state remains *seconded*. In the *voting* state, vote processes are followed by counting until the time for voting is over or all the participants have voted. If the number of *yes* votes is greater than the ceiling, then the protocol terminates successfully in an *agreed* state otherwise the motion is *rejected*. There are other protocols for multilateral negotiation which provide interaction between one to many parties and many-to-many parties, as in protocols for communication in channels.

Figure 4.4 shows the state transition diagram proposed in [OSM Saarl 1998] for multi-lateral negotiation. However it does not illustrate a number of features e.g. the particulars of the compound transitions *amend*, *call* and *vote*, that can succeed or fail. There is no information about which agent can perform which action, so an agent can raise a motion and second its own motion and call a vote. Figure 4.4 is corrected in order to show compound transitions, which are an instance of a protocol given in a circle near that transition. Figure 4.5 defines a more complete protocol than the original version in Figure 4.4.

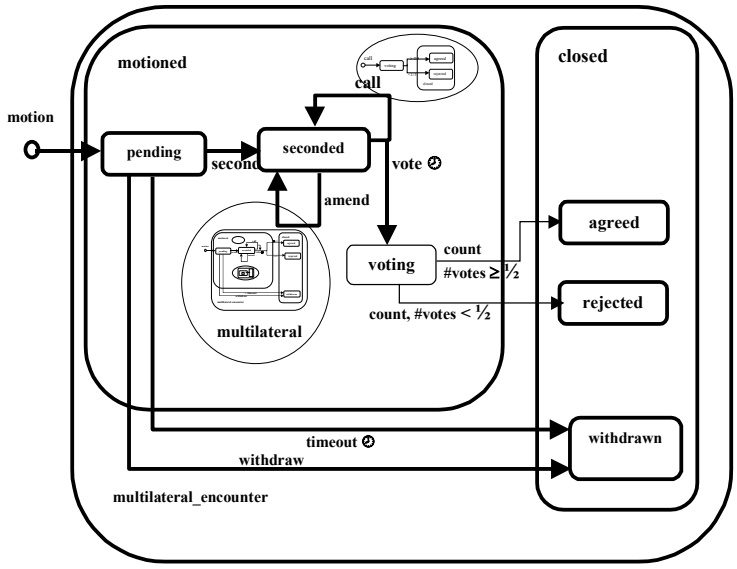


Figure 4.5 Showing compound transitions in multi-lateral protocol

The negotiation process launched at a compound *amend* message follows a multi-lateral protocol where the new subject of negotiation is whether to amend the motion that has been raised in the parent negotiation. Figure 4.6 shows the states and transitions that occur at the *call* transition. If at least two third of the group of agents agree to call a vote, then the call transition in Figure 4.6 succeeds.

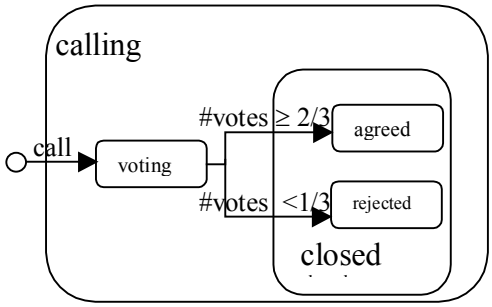


Figure 4.6 The call transition

In Figure 4.5, a motion is accepted if the total numbers of votes counted exceeds at least half the number of participants. Even though Figure 4.5 is an improvement on the original protocol in portraying compound transitions, it contains a number of ambiguities. Representing sub-negotiations in state transition diagrams as above may lead to cluttered figures. Conditions like *#votes* < ½ need to be clarified. Re-initialisation of countdown to voting after successful amendment and how the *voting* state depends on the success of a call are not shown. Thus the resulting state and effect of a sub-process on exiting are not illustrated in the above diagrams. The multi-lateral protocol is represented in ANML to capture all the possible actions and states.

Relation between states and sub-states

A multi-lateral encounter is either *motioned* or *closed*. A *closed* state is in *agreed*, *rejected* or *withdrawn*. The sub-states of *motioned* are *pending*, *seconded* and *voting*.

$$multilateral\_encounter \leftrightarrow \text{one-of}(\{motioned\ ,\ closed\})$$
$$closed \leftrightarrow \text{one-of}(\{agreed\ ,\ rejected\ ,\ withdrawn\})$$
$$motioned \leftrightarrow \text{one-of}(\{pending, seconded, voting\})$$

$$\neg\ multilateral\_encounter \leftrightarrow \text{none-of}(\{motioned\ ,\ closed\})$$
$$\neg\ closed \leftrightarrow \text{none-of}(\{agreed\ ,\ rejected\ ,\ withdrawn\})$$
$$\neg\ motioned \leftrightarrow \text{none-of}(\{pending, seconded, voting\})$$

A multilateral process, *m*, between a set of agents, *G*, terminates in a *closed* state.

$$\neg\ multilateral\_encounter \leftrightarrow\ [G.multilateral\_negotiation_m]\ closed$$

Raising a motion *m* starts a multi-lateral encounter in the *motioned* state, followed by steps for adopting or withdrawing the motion.

$$\neg multilateral\_encounter \leftrightarrow [X.motion_m]\ pending_m(X)$$

The initiator can withdraw its motion *m* or the motion may time out leading to a *withdrawn* state. Otherwise from a *pending* state, a *seconded* state is triggered by another participant seconding the motion *m*.

$$pending_m(X) \leftrightarrow [Y.second_m]\ seconded_m(Y) \vee [timeout]withdrawn_m \vee$$
$$[X.withdraw_m]withdrawn_m \wedge \neg(X{=}Y)$$

$$Y \in G \leftrightarrow (Y.amend_{m1} :: G.multilateral\_negotiation_{m3})$$
$$Y \in G \leftrightarrow (Y.call_{m2} :: G.multilateral\_negotiation_{m2})$$

Proper representation of the multi-lateral protocol requires correctly specifying compound transitions and their conditions. The above two axioms define the processes $amend_{m1}$ and $call_{m2}$ as complex processes each launching a new *multilateral_negotiation*, involving group G with motions m_3 (for an amendment of m with m_1) and m_2 (for a call to vote).

Three transitions are possible from a *seconded* state – a *vote* event, an *amend* of the negotiation subject or a *call* to vote.

$$seconded_m(X) \leftrightarrow (\ [timeout; G.vote_m]\ voting_m(G) \vee$$
$$(\ [Y.amend_{m1}; agreed_{m3}?; reinitialise]\ seconded_{m1}(Y)) \vee \ (\ $$
$$[Y.call_{m2}; agreed_{m2}?; G.vote_m]voting_m(G))) \ \wedge \neg(X{=}Y)$$

In the $seconded_m$ state, when the countdown to voting has elapsed, the group votes on the motion m in the state $voting_m(G)$.

If agent Y wants to amend the subject to replace the motion m with m_1, a multi-lateral sub-process with motion m_3 is launched. If the sub-process $Y.amend_{m1}$ **::** $G.multilateral\_negotiation_{m3}$ terminates in an $agreed_{m3}$ state, the subject of negotiation and time are re-initialised in a new $seconded_{m1}(Y)$ state, otherwise there is no change of state i.e. $seconded_m(Y)$.

In the $seconded_m$ state, if agent Y calls for a vote, then a sub-process *call* is forked between G following a $Y.call_{m2}$ **::** $G.multilateral\_negotiation_{m2}$ process. The motion *m2* is whether to vote immediately on the motion m.

In a $voting_m$ state, when the time for voting is over or all the participants have voted, a *count* process checks whether the number of *yes* votes is greater than the required ceiling and if so the motion is *agreed*.

$$voting_m(G) \leftrightarrow [count; (\#votes \geq 1/2)?]\ agreed_m \vee [count; (\#votes < 1/2)?]\ rejected_m$$

4.5 Scenario for Multilateral Negotiation

Consider a scenario between agents representing people on a management board as an example of applying the above multi-lateral protocol. The management board of Hexa-Decimal company are discussing their company's expansion. The agents in the group are *Ann*, *Betty*, *Chris*, *Duncan*, *Ed*, *Frank*, *Greg*, *John*, *Ken*, *Lily*, *Mary* and *Perry*. The subject of negotiation is the company's expansion project.

Agent *Duncan* starts the negotiation by raising a motion, *mot*, with the subject *X* being the building a new branch for the company. *Mary* seconds *Duncan*'s motion and the state is $seconded_{mot}(Mary)$ with a countdown to the $voting_{mot}$ state. *Chris* calls for a vote but his call fails and the state remains *motioned* and $seconded_{mot}(Mary)$. *Perry* then proposes an amendment to the current motion with subject, mot_1: to renovate and expand their current headquarters. This leads to a sub-negotiation complying to a multilateral protocol and the subject mot_3 is replacing *mot* by mot_1. The multilateral sub-negotiation results in an $agreed_{mot3}$ state, the amendment, mot_1, is accepted as the new motion and countdown to a vote is re-initialised. When the countdown is over, an internal vote event triggers the $voting_{mot1}$ state. Participants either vote *no*, *yes* or *abstain*. With seven *yes* votes, three *no* votes and two *abstain*, the motion, mot_1, is accepted to renovate and expand Hexa-Decimal Company's headquarters.

Agent	Action/event	Resulting Negotiation state
Duncan	$Duncan.motion_{mot}$ Duncan raises a motion, *mot*, with subject to build a new branch of the company	$pending_{mot}(Duncan)$ The state is *pending* on subject *mot*
Mary	$Mary.second_{mot}$ Mary seconds Duncan's motion on the current	$seconded_{mot}(Mary)$ The countdown is started to

	subject.	5 minutes before voting.
Chris	*Chris*.$call_{mot}$ *Chris* calls for a vote. This is a compound transition, which launches a vote for whether voting on the motion, *mot*, can start without waiting for the end of the countdown. Only 4 persons are in favour of the call which is less than the required 2/3. The call fails.	$seconded_{mot}$(*Mary*) The call compound transition yields a rejected state and the state of the multi-lateral negotiation remains $seconded_{mot}$(*Mary*) as before.
Perry	*Perry*.$amend_{mot1}$ *Perry* proposes an amendment to the subject of the motion, *mot*, to a new motion, mot_1: to renovate and expand current headquarters. $amend_{mot1}$ is a compound transition that launches a multilateral negotiation, with subject, mot_3: to replace *mot* with mot_1 from *Perry*. *Ann* seconds the motion mot_3. The state of the sub-encounter is $seconded_{mot3}$(*Ann*). After 5 minutes countdown to a vote, a voting takes place on whether to accept mot_3. The state of the sub-process is $voting_{mot3}$. Ten agents vote *yes* and mot_3 is agreed upon.	The sub-negotiation for the amendment is $agreed_{mot3}$ and the motion in the main negotiation is changed to mot_1 and the countdown re-initialised. The state after a successful compound amend transition is $seconded_{mot1}$(Perry).
Group	$vote_{mot1}$ Voting on the motion mot_1 is started after the countdown is over with the *vote* transition. Participants can either vote *yes*, *no* or *abstain*.	$voting_{mot1}$(*G*) The current state is *voting*.
The compound transition *count*	*count* The number of votes regarding adopting the motion is counted. The motion is accepted if the number of *yes* votes exceeds half the number of agents, otherwise it fails. There are 7 *yes* votes, 3 *no* votes, and 2 *abstain*.	*agreed* on mot_1 More than the required ½ of the board has agreed to mot_1. A quorum has been reached and the motion: to renovate and expand current headquarters is accepted.

Scenario 4.4 A multilateral negotiation for the expansion of a company

4.6 Promissory Negotiation

The protocol for a promissory negotiation shown in Figure 4.7 specifies how promises are made and fulfilled between two agents. Although the protocol is explained in terms of two agents, *X* and *Y*, each agent can be considered as a group of agents. *X* and *Y* may also be agents in a larger group involved in the negotiation. A promissory negotiation is entered by an agent, *X*, making either a promise or a commitment to agent *Y* about a subject. The agent *Y* has a right to call on *X*'s promise before *Y*'s right expires. Agent *X* then has an obligation to fulfil its promise or commitment and until it does so, it is regarded as a *pending* obligation on *X*. From the *pending* state, either *X* fulfils its obligation through a *fulfil* compound transition or the *pending* state times out leading to an *overdue* sub-state. *fulfil* is a compound transition that spawns a bilateral sub-negotiation between agent *X* and *Y* with *proposed* as initial state. The result of the bilateral sub-negotiation determines the end state of its parent promissory process. An *agreed* bilateral sub-process terminates the promissory negotiation in a *fulfilled* state, otherwise the promissory interaction is *rejected*. An agent may invoke a *waive* compound transition, which launches a bilateral negotiation.

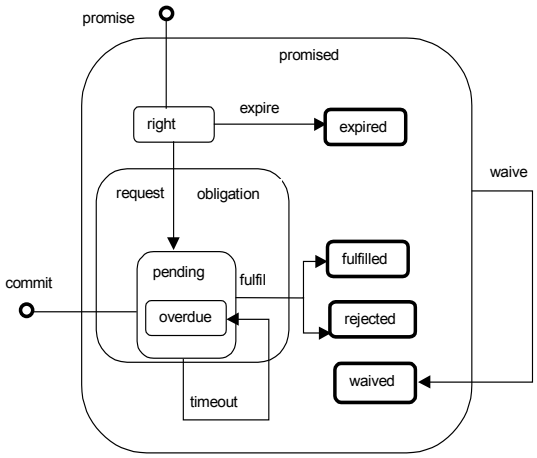


Figure 4.7 Original version of promissory negotiation

A promissory negotiation can be used in contract execution where an agent promises or commits to supplying a service and is called upon to fulfil its obligations. The initial version of a promissory protocol in Figure 4.7, [OSM Saarl 1998], does not illustrate that *fulfil* and *waive* are compound transitions forking bilateral negotiations. Parameterisation of a process with agents and explicit *closed* states are not shown.

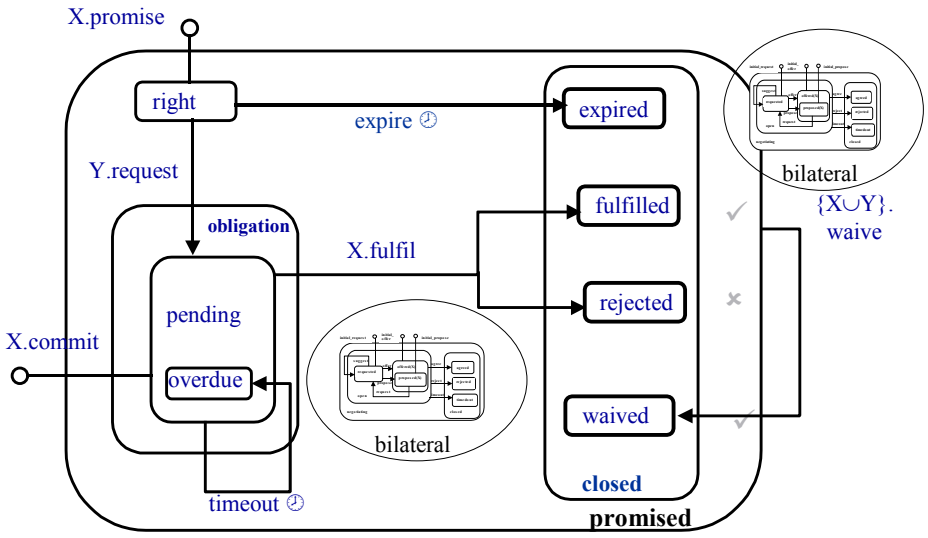


Figure 4.8 Promissory diagram showing compound transitions

Figure 4.8 portrays an improved state transition diagram showing the missing features. However, Figure 4.8 does not express the relation between the state of a sub-process and the state of its parent process. For example, a *rejected* or *timedout* bilateral sub-negotiation forces *fulfil* into a *rejected* state. It is hard to show parameterisation of the states in a statechart when there are several processes leading to the same state. For example in Figure 4.8, *Y* can make a request to trigger the *pending*(*Y*) state, but either agent *X* or *Y* can enter the negotiation with a *commit* yielding *pending*(*Y*) or *pending*(*X*) state. In addition, since it is not realistic to invoke a *waive* from a *closed* state, a *waive* should not be possible from a *promised* state. Therefore the states *right* and *pending* are added as sub-states of *open*, from which *waive* is possible.

The initial state of a promissory negotiation is ¬*promised* and the end state is *closed*. An example of a path in a promissory process between *X* and *Y* is *Y.promise*; *X.request*; *Y.fulfil* - *Y* makes a promise that *X* requests *Y* to fulfil. Another path may be *X.commit; X.fulfil* - *X* makes a commitment and fulfils it. A successful promissory negotiation results in a *fulfilled* state. The ANML theory Theory 4.2 disregards the *obligation* state in Figure 4.8 as it is redundant.

$promised \leftrightarrow$ one-of($\{open, closed\}$)

$open \leftrightarrow$ one-of($\{right, pending\}$)

$closed \leftrightarrow$ one-of($\{expired , fulfilled, rejected , waived\}$)

$overdue(X) \rightarrow pending(X)$

$\neg promised \leftrightarrow$ none-of($\{open, closed\}$)

$\neg open \leftrightarrow$ none-of($\{right, pending\}$)

$\neg closed \leftrightarrow$ none-of($\{expired , fulfilled, rejected , waived\}$)

$\neg promised \leftrightarrow [\{X, Y\}.promissory\text{-}negotiation_p]\ closed$

$\neg promised \leftrightarrow [X.promise_p]\ right_p(X) \vee [X.commit_p]\ (pending_p(X) \wedge \neg overdue)$

$right_p(X) \leftrightarrow (\ [expire]\ expired \vee\ [Y.request_p]\ (pending_p(X) \wedge \neg overdue))$

$\wedge \neg(X = Y)$

$X.waive_p :: \{X,Y\}.bilateral\_process_{p1}$

$Y.fulfil_p :: \{X,Y\}.bilateral\_process_{p2}$

$open \leftrightarrow [\ X.waive_p\ ;\ agreed_{p1}?]\ waived_p \wedge \neg(X = Y)$

$pending_p(Y) \leftrightarrow [timeout]\ overdue(Y) \vee [Y.fulfil_p;\ agreed_{p2}?]\ fulfilled_p \vee$

$[(\ rejected_{p2} \vee timedout_{p2})?]\ rejected_p$

Theory 4.2 Promissory protocol in ANML

Let process p be an instance of a promissory negotiation between two agents X and Y starting in a $\neg promised$ state and terminating in a *closed* state. The negotiation is entered by agent X making either a *promise* or a *commit*. From the $right_p(X)$ state, Y makes a $request_p$ for fulfilment triggering $pending_p(Y)$ or the negotiation can expire. If an agent Y does not fulfil its obligation in the $pending_p(Y)$ state before a timeout, then the negotiation becomes $overdue(Y)$. $Y.fulfil_p$ is a compound transition launching a bilateral negotiation between X and Y with subject p_2 to fulfil p. Similarly, a $waive_p$ transition engages both agents in a bilateral negotiation with subject p_1 to waive p.

4.7 Scenario for Promissory negotiation

Two scenarios of negotiations complying to a promissory protocol are presented. Both of them implicitly include concepts in fulfilling a contract.

4.7.1 Redeeming a voucher

The first scenario involves the agents *Oliver* and *Beach* Hols. *Oliver* won a competition awarding £500 off a holiday with the travel agent *Beach* Hols. Both agents engage in a promissory negotiation with subject, *voucher* (£500 off a holiday, before November, flights and hotel). *Beach* Hols starts the negotiation with a promise to execute the subject *voucher*. *Oliver* requests for his prize and *Beach* Hols fulfils the contract by launching a bilateral negotiation such as in Scenario 4.3 with initial state *proposed*(*beach*) on subject *voucher*. The promissory negotiation ends in a *fulfilled* state with *Oliver* obtaining his holiday.

Agent belonging to	Action	Resulting Negotiation state
Beach Hols	$beach.promise_{voucher}$ Subject *voucher*: £500 off holiday, before November	$right_{voucher}(beach)$ *Oliver* has a right to ask £500 off a holiday before November
Oliver	$oliver.request_{voucher}$ Subject *voucher*: £500 off holiday to Mauritius before November	$pending_{voucher}(beach)$ The state becomes *pending* waiting for execution of the subject *voucher*.
Beach Hols	$beach.fulfil_{voucher}$ This launches a subsidiary bilateral negotiation with subject *voucher*, which successfully terminates with an agreement *S1*.	$fulfilled_{voucher}$ *S1= Beach* Hols sells holiday to Mauritius, 10/08- 24/08, flights + hotel, £1600 (£500 off included) to *Oliver*.

Scenario 4.5 Promissory negotiation for redeeming a voucher

4.7.2 Fulfilling a Commitment

The second scenario involves *Cindy*'s agent requesting *Lake* College to fulfil its obligation to provide her with accommodation. The two agents involved are *Cindy* and *Lake* and the negotiation subject is college accommodation for *Cindy*. They negotiate according to a promissory protocol, started in a *pending* state through *Lake* making a commitment on the negotiation subject. When the *pending* state times out, the negotiation becomes *overdue* until finally *Lake* fulfils its obligation. The bilateral

sub-process spawned by *fulfil* terminates with a rejection from *Cindy*, leading to a *rejected* promissory negotiation.

Agent	Action/event	Resulting Negotiation state
Lake	*Lake.commit*$_S$ Subject *S*: provide accommodation to *Cindy*	*pending*$_S$(*Lake*) Subject: *S*
event	*timeout* A *timeout* occurs after 2 weeks.	*overdue* Subject: *S* *Lake* College has an *overdue* commitment.
Lake	*Lake.fulfil*$_S$ The *fulfil* transition launches a bilateral negotiation, with initial state *proposed*(*Lake*) and subject *S1*: a double room 1 mile from college	The state of the subsidiary bilateral negotiation is *proposed*(*Lake*) with *S1*. The state of the promissory negotiation is *overdue*(*Lake*)
Cindy in sub-process	*Cindy* makes a *request* with *S2*: single room at least 100m from college	The bilateral negotiation's state is *requested*(*Cindy*) with *S2*. The state of the promissory negotiation is *overdue*(*Lake*)
Lake in sub-process	*Lake* sends an *offer* with *S3*: single room, 1 hour from college	The bilateral negotiation's state is *offered*(*Lake*) with *S3*
Cindy in sub-process	*Cindy* sends a *reject* ending the bilateral negotiation in a *rejected*$_{S1}$ state. Control returns to the parent promissory negotiation.	Both the bilateral negotiation and promissory negotiation unsuccessfully terminate in a *rejected* state.

Scenario 4.6 Promissory negotiation for college accommodation

4.8 Auction Protocols

Auctions are a popular type of negotiation. There are various forms of auctions with different protocols for bidding and disclosure such as English, Dutch, sealed bid and

Vickrey auctions. In some auctions, the strategies of the parties are known and the subject involves price. Bargaining is usually restricted to price and the seller's strategy to assign awards is made known. Auctions are thus usually simpler to implement than more interactive protocols. There are currently automated, semi-automated and manual auctions online.

4.8.1 English Auction

In a typical English auction, an auctioneer posts an item that he wants to sell to the highest bidder. The bidders bid against themselves and after each bid, the auctioneer calls for higher bids at most thrice. If after calling for three times, there are no more bids, then the item is sold to the last bidder. Figure 4.9 is a statechart for an English auction and its ANML theory is given in Theory 4.3. Figure 4.9 contains an error in that it allows an auctioneer to perform a *going3rd* without *going2nd*.

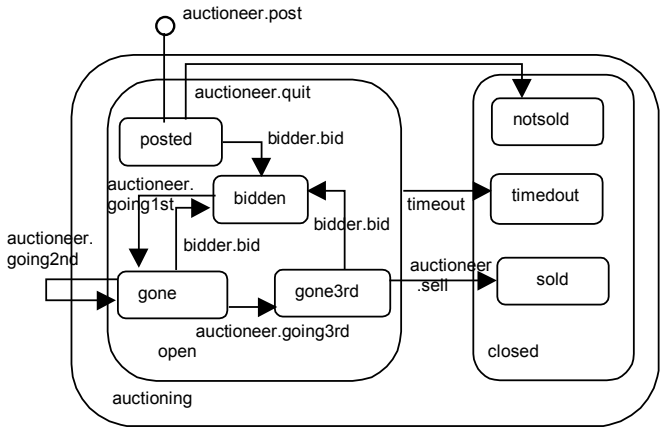


Figure 4.9 Statechart of an English Auction

auctioning ↔ one-of({*open* , *closed*})

open ↔ one-of({*posted*, *bidden*, *gone*, *gone2nd*, *gone3rd*]

closed ↔ one-of({*notsold, timedout*, *sold*})

¬*auctioning* ↔ none-of({*open* , *closed*})

¬*open* ↔ none-of({*posted*, *bidden*, *gone*, *gone2nd*, *gone3rd*]

¬*closed* ↔ none-of({*notsold, timedout*, *sold*})

¬*auctioning* ↔ [({*Auctioneer*:*auctioneer*} ∪ *Group*: *bidders*).*english-auction*$_e$]
closed

¬*auctioning* ↔ [*Auctioneer*:*auctioneer. post*] *posted*(*Auctioneer*)

$posted(Auctioneer) \leftrightarrow [Auctioneer.quit]notsold \vee [Bidder: bidder.bid]$
$((Bidder \in Group) \rightarrow bidden(Bidder))$
$bidden(Bidder) \leftrightarrow [Auctioneer:auctioneer.going1st]gone(Bidder) \vee [Bidder2:$
$bidder.bid]\ ((Bidder2 \in Group: bidders) \rightarrow bidden(Bidder2))$
$gone(Bidder) \leftrightarrow [Auctioneer:auctioneer.going2nd]gone2nd(Bidder) \vee [Bidder2:$
$bidder.bid]\ ((Bidder2 \in Group: bidders) \rightarrow bidden(Bidder2))$
$gone2nd(Bidder) \leftrightarrow [Auctioneer:auctioneer.going3rd]gone3rd(Bidder) \vee$
$[Bidder2: bidder.bid]\ ((Bidder2 \in Group: bidders) \rightarrow bidden(Bidder2))$
$gone3rd(Bidder) \leftrightarrow [Auctioneer:auctioneer.sell]\ sold \vee [Bidder2: bidder.bid]$
$((Bidder2 \in Group: bidders) \rightarrow bidden(Bidder2))$
$auctioning \leftrightarrow [timeout]timedout \vee [gone3rd(Bidder)?;$
$Auctioneer:auctioneer.sell]\ sold$

Theory 4.3 English Auction in ANML

4.8.2 Scenario of an English Auction

Scenario 4.7 is an English auction between an auctioneer called *Louvres* and registered bidders *Tate*, *Pompidou*, *Orsay*, *Christie*, *Sotheby*, *National-UK*, *Uffizi*, *Washington-art*, *Ontario-art*, *Pushkin-gallery*, *Museo-prado*, *Sistine*, *Mary*, *Ned*, *Oscar* and *Perry*. The subject of the auction is the sale of the Mona Lisa painting. The auctioneer starts the auction by posting the painting at 1 billion pounds.

Agent	Action/event	Resulting Negotiation state
Louvres	*Louvres.post* Subject *S*: Mona-Lisa painting starting price 1 billion	*posted*(*Louvres*) The Mona-Lisa painting posted at 1 billion.
Christie	*Christie.bid* Subject: Mona-Lisa at 10 billion	*bidden*(*Christie*) Subject: Mona-Lisa at 10 billion The countdown to going1st transition is started.
Sotheby	*Sotheby.bid* Subject: Mona-Lisa at 12 billion	*bidden*(*Sotheby*) Subject: Mona-Lisa at 12 billion The countdown to going1st is restarted

Pushkin	*Pushkin.bid* Subject: Mona-Lisa at 50 billion	*bidden*(*Pushkin*) Subject: Mona-Lisa at 50 billion Countdown to *going1st* is restarted.
Louvres	*Louvres.going1st* Subject: Mona-Lisa at 50 billion The countdown to *going1st* ends and the auctioneer calls for another *bid* for the first time.	*gone*(*Pushkin*) Subject: Mona-Lisa at 50 billion The countdown to *going2nd* is started.
Louvres	*Louvres.going2nd* Subject: Mona-Lisa at 50 billion The countdown to *going2nd* ends and the auctioneer calls for a *bid* for the second time.	*gone2nd*(*Pushkin*) Subject: Mona-Lisa at 50 billion The countdown to *going3rd* is started.
Louvres	*Louvres.going3rd* Subject: Mona-Lisa at 50 billion The countdown to *going3rd* ends and the auctioneer calls for a *bid* for the third time.	*gone3rd*(*Pushkin*) Subject: Mona-Lisa at 50 billion The countdown to being *sold* is started.
Uffizi	*Uffizi.bid* Subject: Mona-Lisa at 100 billion *Uffizi* makes a *bid* at 100 billion pounds leading back to *bidden*	*bidden*(*Uffizi*) Subject: Mona-Lisa at 100 billion The countdown to *going1st* is started
…	…	…
Louvres	*Louvres.going3rd* Subject: Mona-Lisa at 500 billion The countdown to *going3rd* ends and the auctioneer calls for a *bid* for the third time.	*gone3rd*(*Tate*) Subject: Mona-Lisa at 500 billion The countdown to being *sold* is started.
Louvres	*Louvres.sold* The countdown to *sold* ends and the auctioneer sells the painting to *Tate*.	*sold*

Scenario 4.7 English Auction for the sale of a painting

4.8.3 Dutch Auction

In a Dutch auction, the auctioneer starts the auction at a high price and progressively lowers the price until either a participant bids or the auctioneer quits. An auctioneer usually has a reserve price and he withdraws from the auction if there are no bids at that limit. The item is sold to the first bidder. Dutch auctions are used to sell quantities of flowers in Holland. A Dutch auction protocol can avoid contention amongst buyers by accepting the first bid and disregarding other bids.

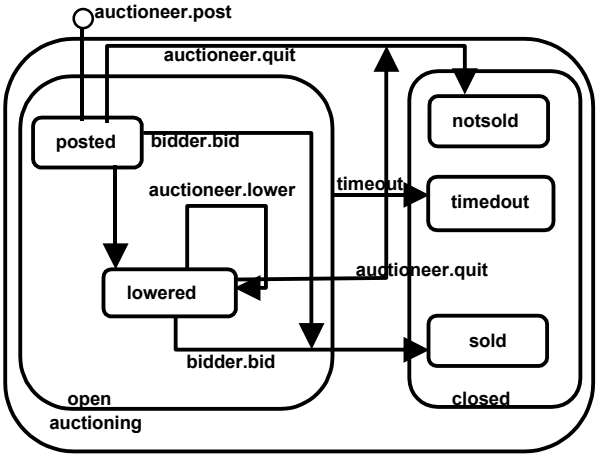


Figure 4.10 Dutch Auction state diagram

Theory 4.4 expresses the protocol for a Dutch auction in ANML.

$auctioning \leftrightarrow$ one-of($\{open, closed\}$)

$open \leftrightarrow$ one-of($\{posted, lowered\}$)

$closed \leftrightarrow$ one-of($\{notsold, timedout, sold\}$)

$\neg auctioning \leftrightarrow$ none-of($\{open, closed\}$)

$\neg open \leftrightarrow$ none-of($\{posted, lowered\}$)

$\neg closed \leftrightarrow$ none-of($\{notsold, timedout, sold\}$)

$\neg auctioning \leftrightarrow [(\ \{Auctioneer{:}auctioneer\} \cup Group{:}\ bidders).\ dutch\text{-}auction_d]$
$closed$

$\neg auctioning \leftrightarrow [Auctioneer{:}auctioneer\ .\ post]\ posted(Auctioneer)$

(*posted*(*Auctioneer*) ∨ *lowered*(*Auctioneer*)) ↔ [*Auctioneer.quit*] *notsold* ∨
[*Auctioneer.lower*]*lowered*(*Auctioneer*) ∨ [*Bidder*: *bidder*. *bid*]
((*Bidder*∈*Group*: *bidders*)→ *sold*)

open ↔ [*timeout*]*timedout* ∨ [*Auctioneer*: *auctioneer.quit*] *notsold* ∨ [*Bidder*:
bidder.bid] ((*Bidder* ∈ *Group*: *bidders*) → *sold*)

Theory 4.4 ANML Protocol for Dutch Auction

4.8.4 Sealed Bid Auction

In a sealed bid auction, an auctioneer advertises a service to a group of bidders. The bidders send their bids to the auctioneer who decides which bid to accept. The auctioneer can quit the auction if there are no bids or if he refuses all bids.

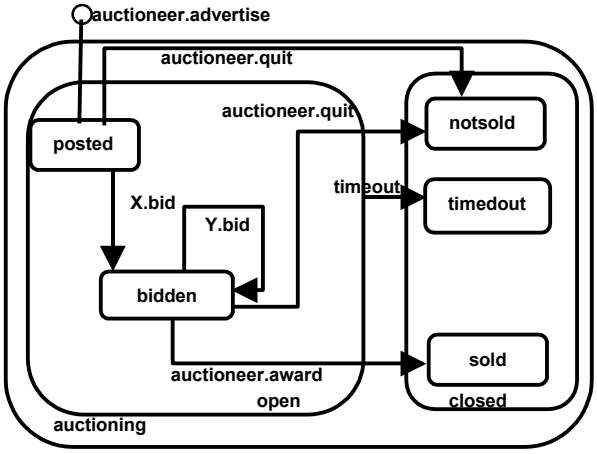


Figure 4.11 State diagram for Sealed Bid Auctioning protocol

auctioning ↔ one-of({*open*, *closed*})

closed ↔ one-of({*notsold*, *timedout*, *sold*})

open ↔ one-of({*posted*, *bidden*})

¬*auctioning* ↔ none-of({*open*, *closed*})

¬*closed* ↔ none-of({*notsold*, *timedout*, *sold*})

¬*open* ↔ none-of({*posted*, *bidden*})

¬*auctioning* ↔ [({*Auctioneer*:*auctioneer*} ∪ *Group*: *bidders*). *sealed-bid-auction*$_{sb}$] *closed*

$\neg auctioning \leftrightarrow [Auctioneer{:}auctioneer.advertise]\ posted(Auctioneer)$

$posted(Auctioneer) \leftrightarrow [BidderX{:}\ bidder.bid]\ ((BidderX \in Group{:}\ bidders) \rightarrow bidden) \vee [Auctioneer.quit]\ notsold$

$bidden \leftrightarrow [BidderY{:}\ bidder.bid]\ ((BidderY \in Group{:}\ bidders) \rightarrow bidden) \vee [Auctioneer{:}auctioneer.award]\ sold \vee [Auctioneer{:}auctioneer.quit]\ notsold$

$open \leftrightarrow [timeout]timedout \vee [Auctioneer{:}auctioneer.quit]\ notsold \vee [bidden?;\ Auctioneer{:}auctioneer.award]\ sold$

Theory 4.5 A sealed-bid auction in ANML

4.9 Fish-Market Auction

Rodriguez and al. [1998a] provide mechanisms for negotiation in the context of a fish-market auction. In a market in Barcelona, boxes of fish are sold in a fish-market auction. The market acts as an institution that enforces the rules between buyers and sellers in an auction. It establishes conventions for the types of goods sold, interactions between participants, their rights and obligations. Several scenes occur simultaneously at different locations. Before an auction, the fishermen deliver fish to the market at the sellers' admission scene while the buyers register at the buyers' admission scene. In the online Fish-Market platform, seller agents register their goods with a seller admitter agent and buyers register with a buyer admitter agent. The auction follows a downward bidding protocol, where buyers bid for boxes of fish that are presented by an auctioneer who calls prices in descending order. After an auction in the Barcelona market, a successful bidder collects its box of fish through a buyer's settlement scene and a seller collects its payments for its lot at a sellers' settlement scene. In the online version, seller agents obtain their payment from a seller manager and buyer agents pay through a credit line that is managed by a seller manager. Figure 4.12 illustrates the Fish Market protocol as a statechart but it contains some errors since it does not show an auctioneer raising the price during a collision. One cannot deduce that the *delivered* and *registered* states can coexist. Similarly *committing* should be a sub-state of *sold* since the box of fish remains *sold* when committing or paying for it. Parameterisation of states with an agent is not shown because of problems mentioned earlier when several processes lead to the

same state. In addition, it is difficult to show the case of colliding bids as two simultaneous events occurring in one state.

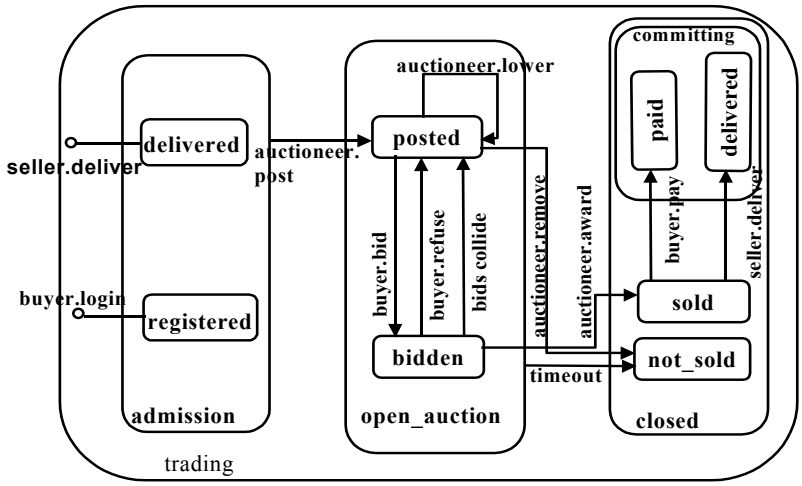


Figure 4.12 FishMarket Auction Protocol

The Fish-Market protocol can be expressed as an ANML theory. Negotiation where bidders, sellers and auctioneer engage in an auction constitutes the interesting aspect.

trading ↔ one-of({*admission*, *open_auction*, *closed*, *sold*})

admission ↔ *delivered* ∨ *registered*

closed ↔ one-of({*ended*, *not-sold*})

committing ↔ one-of({*paid*, *collected*})

sold ↔ one-of({*awarded*, *committing*})

open_auction ↔ one-of({*posted*, *bidden*, *simultaneously_bidden*})

¬*trading* ↔ none-of({*admission*, *open_auction*, *closed*, *sold*})

¬ *closed* ↔ none-of({*ended*, *not-sold*})

¬ *committing* ↔ none-of({*paid*, *collected*})

¬*sold* ↔ none-of({*awarded*, *committing*})

¬*open_auction* ↔ none-of({*posted*, *bidden*, *simultaneously_bidden*})

¬*trading* ↔ [*Group-of-agents*: *participants*. *FishMarket*$_f$] *closed*

The admission phase to the Fish market is given by entry to the *admission* state.

$\neg trading \leftrightarrow$ [*Seller*:*seller.deliver*] *delivered*(*Seller*) $\vee$ [*Buyer*:*buyer.login*] *registered*(*Buyer*)

A *committing* phase is enabled by a sale from an *open* auction. A *closed* state is triggered by a *timeout*, an auctioneer's withdrawal or the end of a commitment. If at least two different bidders bid simultaneously then the state triggers to *simultaneously_bidden*.

delivered(*Seller*) $\leftrightarrow$ [*Auctioneer*:*auctioneer.post*] *posted*(*Auctioneer*).

posted(*Auctioneer*) $\leftrightarrow$ [*Autioneer.quit*] *not-sold* $\vee$ [*Autioneer.lower*] *posted*(*Auctioneer*) $\vee$ ([*Group*: *buyers.bid*] ((*Buyer1*:*buyer* $\in$ *Group*:*buyers* $\wedge$ *Buyer2*:*buyer* $\in$ *Group*:*buyers*) $\rightarrow$ *simultaneously_bidden*(*Group*)) $\vee$ [*Buyer*1:*buyer.bid* $\wedge$ $\neg$*Buyer*2:*buyer.bid*] *bidden*(*Buyer1*) $\wedge$ $\neg$(*Buyer1=Buyer2*).

bidden(*Buyer1*) $\leftrightarrow$ [*Auctioneer*:*auctioneer.award*] *awarded*(*Buyer1*)

From a *simultaneously_bidden*(*Group*) state, either there is a tie-break process where one agent wins or a collision message from the auctioneer and the price is incremented.

simultaneously_bidden(*Group*) $\leftrightarrow$ [({*Auctioneer*:*auctioneer*} $\cup$ *Group*). *tie_break*] *bidden*(*BuyerX*) $\vee$ [*Auctioneer*:*auctioneer.collision*; *Auctioneer.increase-price*]*posted*(*Auctioneer*).

open_auction $\leftrightarrow$ [*timeout* $\cup$ *Auctioneer*:*auctioneer.quit*]*not_sold* $\vee$ [*bidden*(*Buyer1*)? *Auctioneer*:*auctioneer.award*] *awarded*(*Buyer1*)

In a *sold* state, a successful bidder respects its commitments by paying for the bought boxes of fish and the seller can collect its money.

awarded (*Buyer1*)$\leftrightarrow$ [*Buyer1.pay*] *paid*(*Buyer1*) $\vee$ [*Seller*:*seller.collect*] *collected*(*Seller*)

paid(*Buyer1*) $\leftrightarrow$ [*Seller*:*seller.collect*] *ended*

collected(*Seller*) ↔ [*Buyer*:*buyer.pay*] *ended*

4.10 Summary

This chapter has shown the application of ANML in representing protocols for negotiation. In addition, these protocols are expressed in statecharts and illustrated through scenarios. Expanding a statechart to include compound transactions yields complex and cluttered diagrams. For example, the expanded bilateral negotiation statechart in Figure 4.3 is not easy to understand. A statechart of a protocol may also contain errors and ambiguities. In a statechart that consists of both non-iterative and iterative processes leading to the same state, parameterisation of states and actions with agents cannot be correctly expressed. This is also the case when there are several processes leading to the same state. Expressing a protocol in ANML concisely and intuitively captures all possible transitions and states including launching subsidiary protocols. ANML facilitates the detection and controls infinite regression of sub-negotiations. The history of a negotiation can be accessed by analysing the paths and states that have occurred until the current state. The protocols in this chapter can be extended or other protocols similarly constructed using ANML e.g. mixed many party negotiations (one-to-many/many-to-many channels). There have been several iterations in designing these protocols by repeatedly fixing the errors that come to light when expressing the statecharts in ANML. The next chapter shows the verification aspect of our framework by applying it to expressing existing protocols.

5 Verification of Protocols

5.1 Introduction

This chapter verifies existing protocols for interaction between agents and presents corrected versions in ANML. Agents involved in the same negotiation comply with a common protocol to coordinate meaningfully with each other. Ambiguities and errors in a protocol can lead to misunderstandings between the agents about the history, the current state and possible future actions in a negotiation. A correct and clear protocol, with all states well-defined, allows all agents in a group to share the same public beliefs about a conversation.

The original specification of the bilateral protocol [OSM Saarl 1998], introduced in the last chapter, is checked for errors and imprecision leading to undefined states of negotiation, and corrected. The corrections show how to obtain a bilateral protocol in ANML where all states are defined at all times. Similar verifications are performed on several interaction protocols proposed in AUML, [Odell and al 2000]. By translating an AUML protocol into a path in ANML, errors are highlighted, (some inherent to the original modeling language), [Paurobally and Cunningham 2002a]. For each verified AUML protocol, its corresponding ANML theory is specified as a more complete protocol. Then we compare Petri-Nets with ANML. First a survey of Petri-nets and its properties are discussed [Murata 1989]. We then analyse agent protocols in Petri-Nets, [Petri 1966], and coloured Petri-nets. The Petri net approach is compared to ANML for modeling agent interaction protocols.

A protocol can be computationally interpreted from a state transition, a statechart, an AUML diagram or an ANML theory. The agents may be encoded with some but not complete knowledge about real life interactions. However as shown in the previous chapter, a state transition diagram or statechart cannot fully capture multi-agent interactions. Likewise, diagrammatic representations such as AUML depends on our understanding of real life conversations and thus assumes an agent to have some implicit beliefs about interactions. Such informal specification methods easily give rise to unexpected states and contradictions about the negotiation state between participants' beliefs, especially so when the participants differ in culture. On the other hand, a protocol expressed as a logical theory removes ambiguity, allows verification, ensures completeness and is better suited for reasoning and planning using AI techniques. The penultimate section of this chapter argues why AUML is inadequate as a language for representing protocols between agents, thereby adding to the justification of ANML as an alternative for providing less ambiguity and more formal method.

5.2 Bilateral Protocol

Figure 5.1 shows the initial version of a bilateral protocol as a state transition diagram taken from [OSM SARL 1998]. Detailed explanation of this protocol can be found in the previous chapter. A group of agents involved in a bilateral negotiation can invoke state transitions to achieve goal states such as an agreement. On reaching an agreement, an engagement process may be launched for drawing up a commitment and for setting up contracts between the agents.

When verifying the bilateral protocol, a number of errors are found. The steps by which the bilateral protocol is corrected through repetitive representation, verification and correction in ANML are shown below.

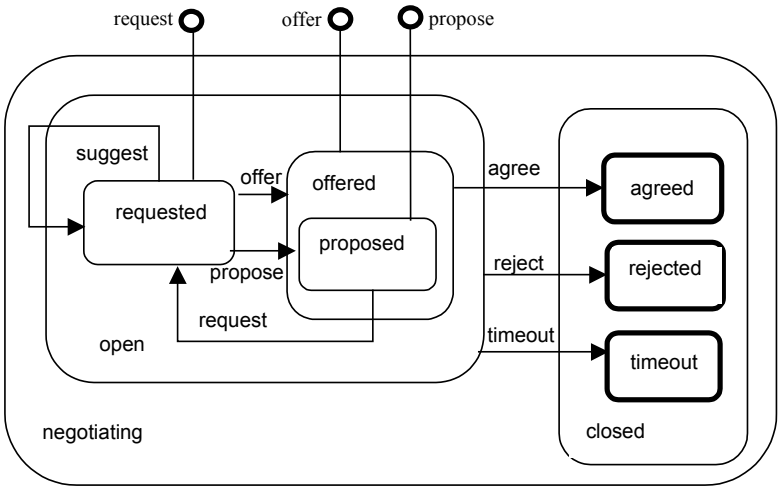


Figure 5.1 Original Bilateral Protocol

5.2.1 Errors in the Original Bilateral Protocol

Analysing Figure 5.1 gives rise to a number of questions: Can a negotiation be *closed* but not *rejected*, *timedout* or *agreed*? Similarly, can the current state be *open* without being in a *requested* or *offered* state. The relations between a parent and its sub-states are explicitly not explained in a state transition diagram, without analysing all possible transitions. Theory 5.1 is an initial version of the protocol expressing both the relation between states and state transitions rules.

$negotiating \rightarrow open \vee closed$

$open \rightarrow requested \vee offered$

$closed \rightarrow agreed \vee rejected \vee timeout$

$proposed \rightarrow offered$

$[request]\ requested$

$[offer]\ offered$

$[propose]\ proposed$

$offered \rightarrow [agree]\ agreed$

$requested \rightarrow [offer]\ offered \vee [propose]\ proposed \vee [suggest]\ requested$

$proposed \rightarrow [request]\ requested$

$open \rightarrow [reject]\ rejected \vee [timeout]\ timedout$

Theory 5.1 Bilateral Protocol Version 1

Incompleteness in the protocol is detected by analysing the truth-values of the states in Theory 5.1. From Theory 5.1, it cannot be inferred that some parent states can only hold if at least one of its sub-states holds. For example, *open* must only be *true* if either of its sub-state is *true*. A single implication between two states means that if a negotiation is in a sub-state then it is in its parent state. A double implication is stronger and should be used to express the dependency of a parent state on its sub-states. (Read as a parent state holds only if its sub-states hold and vice versa). In the case of *proposed* and *offered*, the parent state *offered* can be *true* without being in *proposed*.

Theory 5.1 allows both *requested* and *offered* to hold at the same time or the *closed* state to be in all three *agreed* and *timedout* and *rejected* sub-states. The bilateral protocol needs mutually exclusive sibling states. That is, only one sibling state to be *true* and all other non-parent states *false*. In Theory 5.1, an agent can perform more than one action from a given state e.g. from *requested*, an agent may send 3 messages – an *offer*, a *proposal* and a *suggest* or from *offered* an agent may send both *agree* and *reject* messages. The execution of a process from a source to a target state must be clearly defined along with the constraints for them to fire, specially when more than one action is possible from a state. The *one-of* predicate (see chapter 3) is used to return *true* if and only if exactly one of the states in its given list is *true*, or *false* otherwise.

5.2.2 Theory of Bilateral Negotiation – Corrected Version

Theory 5.2 takes into consideration the above remarks and adds double implications and one-of operators to the rules.

negotiating $\leftrightarrow$ one-of ({ *open, closed* })
open $\leftrightarrow$ one-of ({ *requested, offered* })
closed $\leftrightarrow$ one-of ({ *agreed, rejected, timeout* })
proposed $\rightarrow$ *offered*

[*request*] *requested*

[*offer*] *offered*

[*propose*] *proposed*

offered → [*agree*] *agreed*

requested→ [*offer*] *offered* ∨ [*propose*] *proposed* ∨ [*suggest*] *requested*

proposed → [*request*] *requested*

open → [*reject*] *rejected* ∨ [*timeout*] *timedout*

Theory 5.2 Bilateral Protocol - Version 2

The last seven rules show the entry points and allowed state transitions in a bilateral negotiation. A *requested* state becomes valid after a *request* to enter the negotiation, after a *request* from a *proposed* state or after a *suggest* from a *requested* state. In addition, double implications should be used in state transitions to express those processes that are possible from a source state and none others. For example, only a *suggest*, *offer* or *propose* are possible from a *requested* state in addition to those processes that are possible from its parent state. Double implications in state transitions can also be used to infer the previous states from executing a process and henceforth the history of a negotiation. For example, if the current state is *proposed* then it can be deduced that either there was a *propose* from a *requested* state or the negotiation was started via a *propose*. Theory 5.3 gives revised rules for state transitions with double implications.

[*request*] *requested*

[*offer*] *offered*

[*propose*] *proposed*

offered ↔[*agree*] *agreed*

requested ↔ one-of ({[*offer*] *offered* , [*propose*] *proposed*, [*suggest*] *requested*}).

proposed ↔[*request*] *requested*

open ↔ one-of ({[*reject*] *rejected*, [*timeout*] *timedout*, [*offered* ?*agree*] *agreed*}).

Theory 5.3 Bilateral Protocol – Version 3

There remain ambiguities in the protocol that can be detected in Theory 5.3, but which are not obvious from the corresponding transition diagram. In the current protocol, a negotiation can be started and restarted with a *request*, *offer* and *propose*. During a negotiation, there are no conditions from preventing an agent from restarting

the process by performing one of the entry actions. [*request*]*requested* is *true* in all worlds making *request* possible at all states and likewise for *offer* and *propose*. An agent cannot differentiate between *request*, *offer* or *propose* being entry actions and being state transitions from an *open* sub-state. To remedy this, the names of the three entry actions are changed to *initial_request, initial_offer and initial_propose*. Instead of being undefined, the initial state before entering in a negotiation is made to be ¬*negotiating*.

Conditions about which agent can perform a state transition and who triggered the previous state have to be expressed, for example to prevent an agent agreeing to its own *offer*. This is done through parameterisation of the processes and states with agents. An initiator is an agent that triggered the current state and a respondent is an agent that can respond. Because of the iterations in the bilateral protocol, a state transition diagram cannot correctly show the parameterisation. For example, if agent *X* performs a *request*, the state is *requested*(*X*) from which *Y* can make a *suggest* or *propose*. If *Y* makes a *suggest*, then the state should be *requested*(*Y*) from which *X* can *suggest* or *propose*. It is not possible to show both *requested*(*Y*) and *requested*(*X*) on the same state transition diagram. The same problem applies when different processes can lead to the same state. ANML allows specification that the respondent is not the initiator through parameterisation and negation. The changes that are possible to Figure 5.1 are given in Figure 5.2, although the ANML theory should be referred to for a correct and complete protocol.

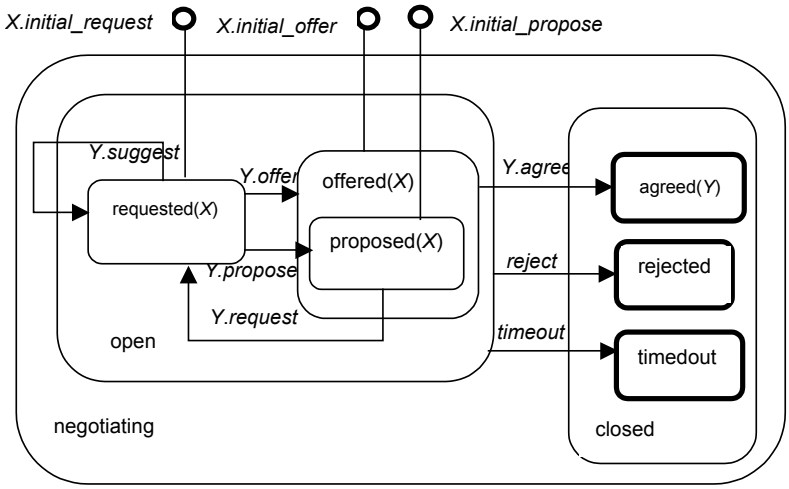


Figure 5.2 Version 2 of state transition diagram for bilateral negotiation

negotiating ↔ one-of ({ *open*, *closed* })

open ↔ one-of ({ *requested*(*X*), *offered*(*X*) })

closed ↔ one-of ({ *agreed*(*Y*), *rejected*, *timeout* })

proposed(*X*) → *offered*(*X*)

¬*negotiating* ↔ one-of ({ [*X.initial_request*]*requested*(*X*), [*X.initial_offer*]
offered(*X*) , [*X.initial_propose*] *proposed*(*X*) }) . (5.4.1)

offered(*X*) ↔ [*Y.agree*] *agreed*(*Y*) ∧ ¬(*X*= *Y*).

requested(*X*)↔ one-of ({ [*Y.offer*] *offered*(*Y*), [*Y.propose*] *proposed*(*Y*),
[*Y.suggest*] *requested*(*Y*)}) ∧ ¬(*X*= *Y*). (5.4.2)

proposed(*X*)) ↔ [*Y.request*] *requested*(*Y*) ∧ ¬(*X*=*Y*).

open ↔ one-of({ [*X.reject*] *rejected*, [*timeout*]*timedout*, [*offered*(*X*))?*Y.agree*]
agreed(*Y*) }) ∧ ¬(*X*=*Y*).

Theory 5.4 Bilateral Negotiation Theory, Version 4

5.2.3 Errors in sub-states

The state *proposed* is a sub-state of *offered* and a negotiation can be *offered* without being *proposed*. When *proposed* is *true*, *offered* must be *true* but not vice versa. Suppose that *X* enters a negotiation with an *initial_offer* triggering *offered*(*X*). According to the protocol, *proposed*(*X*) is undefined and could be either *true* or *false*. In fact, whenever *offered*(*X*) is triggered by an *offer* or *initial_offer*, *proposed*(*X*) is undefined because the theory of the protocol is not complete yet. The condition that *proposed* is *false* after an *offer* or an *initial_offer* is added. If a protocol includes a parent state that can exist without being in one of its sub-states, the value of the sub-states have to be explicitly defined when triggering only the parent state. Rules (5.4.1) and (5.4.2) are replaced with the following two rules (additions in bold):

¬*negotiating* ↔ one-of ({ [*X.initial_request*]*requested*(*X*), [*X.initial_offer*]
(*offered*(*X*) ∧ ¬***proposed*(*X*)**), [*X.initial_propose*] *proposed*(*X*) }) .

requested(*X*)↔ one-of ({ [*Y.offer*] (*offered*(*Y*) ∧ ¬***proposed*(*X*)**), [*Y.propose*]
proposed(*Y*), [*Y.suggest*] *requested*(*Y*)}).

Another problem is that the predicate *one-of* returns *false* if two or more states in its list are *true*. The state ¬*negotiating* is *true* when *one-of*({*open, closed*}) is *false*. This implies that the states *open* and *closed* could both be *true* before entering a negotiation. State ¬*negotiating* must be *true* when *one-of*({*open, closed*}) is *false*, and in addition both *open* and *closed* must be *false*. Likewise the formula *one-of*({*agreed*, *rejected*, *timedout*}) and the state *closed* may be *false* because out of the three states, at least two of them are *true*. Thus the states *agreed* and *rejected* and *open* may all hold when the state is ¬*closed*. A *one-of* predicate in a state transition ensures that at least one action is performed. Yet two actions must not be able to occur simultaneously from the same source state, if not explicitly allowed. For example, *agree* and *reject*. When a parent state is *false*, none of its sub-states should be *true*. The following additional rules use a *none-of* predicate which returns *true* if no elements in its given list are *true*.

¬*negotiating* ↔ none-of ({ *open, closed* })
¬*open* ↔ none-of ({ *requested*(*X*), *offered*(*X*) })
¬*closed* ↔ none-of ({ *agreed*(*Y*), *rejected, timeout* })

The *one-of* predicate can be eliminated in state transitions since this constraint is entailed by the rules about parent and sub-states. Parameterisation are also removed from the relation between states because of the axiom $A(X) \rightarrow A$ in ANML, where A is a state and X is an agent or a group.

Theory 5.5 expresses a bilateral protocol between agents X and Y. It can be seen that the corresponding Figure 5.2 contains errors regarding the parameterisation of actions and states, which cannot be amended in the figure.

Relation between states:

negotiating ↔ one-of ({ *open*, *closed* })
open ↔ one-of ({ *requested*, *offered* })
closed ↔ one-of ({ *agreed*, *rejected*, *timeout* })
proposed(*X*) → *offered*(*X*)
¬*negotiating* ↔ none-of ({ *open, closed* })

$\neg open \leftrightarrow$ none-of ({ *requested*, *offered* })

$\neg closed \leftrightarrow$ none-of ({ *agreed*, *rejected, timeout* })

State transitions:

$\neg negotiating \leftrightarrow [X.initial\_request] requested(X) \vee [X.initial\_offer] (offered(X) \wedge \neg proposed(X)) \vee [X.initial\_propose] proposed(X)$.

$requested(X) \leftrightarrow [Y.offer] (offered(Y) \wedge \neg proposed(X)) \vee [Y.propose] proposed(Y) \vee [Y.suggest] requested(Y) \wedge \neg(X = Y)$.

$offered(X) \leftrightarrow [Y.agree] agreed(Y) \wedge \neg(X = Y)$.

$proposed(X)) \leftrightarrow [Y.request] requested(Y) \wedge \neg(X=Y)$.

$open \leftrightarrow$ **(** $[X.reject] rejected \vee [timeout] timedout \vee [offered(X))?Y.agree] agreed(Y)$ **)** $\wedge \neg(X=Y)$.

Theory 5.5 Final Version of Bilateral Protocol in ANML

5.3 Representing Protocols in AUML

The FIPA agent communication domain, [FIPA 2001], contains specifications about interaction protocols, communicative acts and content languages. FIPA agent communication language (ACL), defined in the communicative act library, is based on speech act theory where messages are actions or communicative acts. Each communicative act is described in both a narrative form and in formal semantics based on modal logic. FIPA uses AUML (Agent Unified Modeling Language), [Odell and al. 2000, FIPA 2001a] to specify interaction protocols between agents. FIPA interaction protocols (IP) range from simple requests for performing or querying an action to setting up contracts and exchanging messages in various types of auctions.

AUML, [Odell and al. 2000; Bauer and al. 2001], is an extension of UML to express agent interaction protocols through UML sequence diagrams. It is a specification technique for interaction protocols with formal and intuitive intended semantics and a user-friendly graphical notation, [Bauer and al. 2001]. However no formal semantics are found in the AUML library specification, [FIPA 2001a]. OCL (Object Constraint Language), [Warmer and Kleppe 1999], can be used as a constraint language but OCL is not a formal method, [Richters and Gogolla 1998] and is designed for object

modeling through UML. FIPA interaction protocols in AUML rely partly on FIPA ACL and its semantics by using a subset of the FIPA communicative acts.

AUML specifies an Interaction Protocol (IP) in the form of a UML sequence diagram with extensions. Agents belong to classes and are assigned roles. An IP (interaction protocol) diagram shows interactions between agents along a timeline.

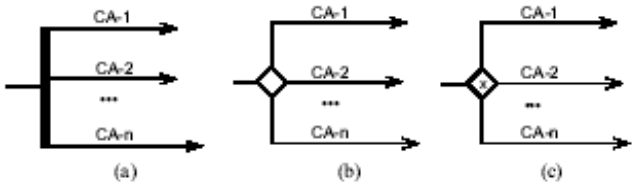


Figure 5.3 Extensions supporting concurrent threads

AUML supports concurrent threads of interaction for sending more than one communicative act. An asynchronous and unnested message is drawn with a stick arrowhead. It shows the sending of a message without yielding control. Figure 5.3 shows three ways of expressing multiple threads. Figure 5.3(a) indicates an *and* communication where all threads *CA-1* to *CA-n* are sent concurrently. Figure 5.3(b) shows a decision box where zero or more CAs (Communicative Acts) may be sent. It indicates an *inclusive or* communication. Figure 5.3(c) indicates an *exclusive or* communication, so that exactly one CA may be sent.

Figure 5.4 is a FIPA AUML IP for the contract net protocol, [Smith 1980]. The name of a protocol is given at the top left of its sequence chart. There are two roles in this protocol: *initiator* or *participant*. An initiator sends a *call-for-proposal* to participants in the contract net process. A participant can respond to the initiator before a given deadline with: a proposal, a refusal, or indicating that it did not understand. In the case of a proposal, an initiator can either accept or reject the proposal. A participant who has received a proposal acceptance eventually informs the initiator about the proposal's execution. A tabbed folder at the upper left indicates that a protocol is a *package* that can be customized for analogous problem domains. A dashed box on the upper right-hand corner declares a protocol as a *template* specification that contains unbound entities, which are bound on instantiation. The rest of this chapter verifies

and shows that protocols in AUML, including Figure 5.4, are vague, contain errors and do not scale to multi-agent interactions.

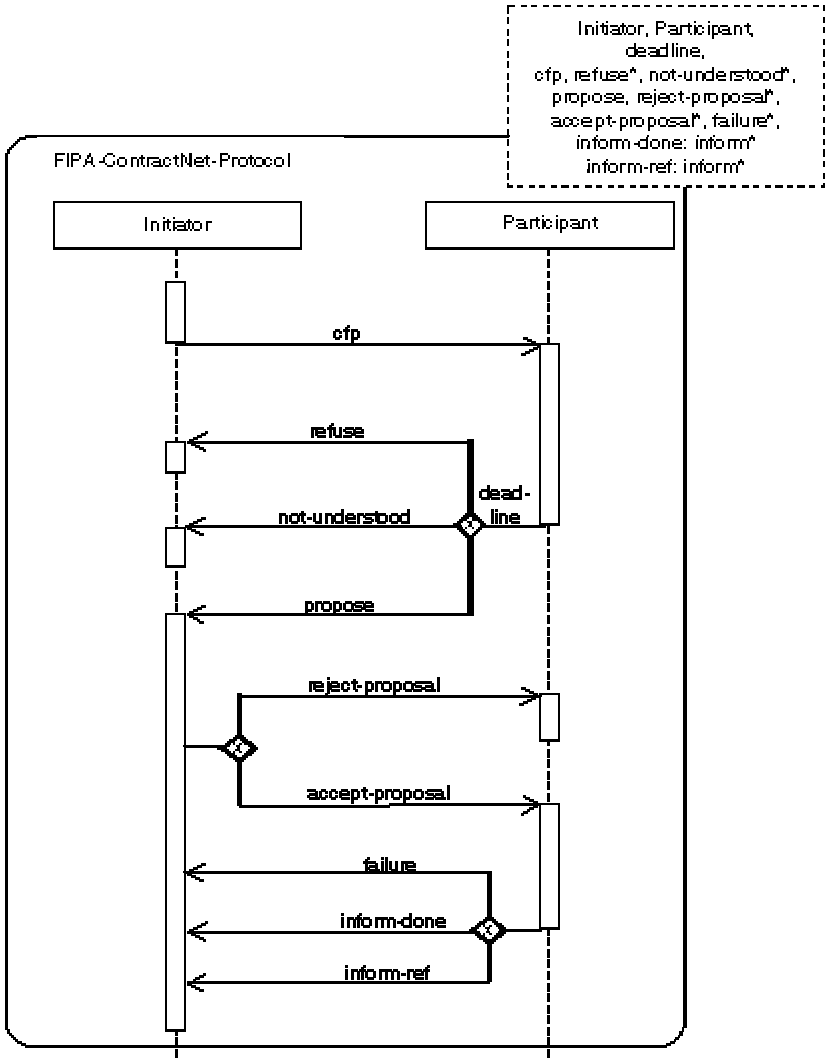


Figure 5.4 A generic IP expressed as a template package, [Odell and al. 2001]

5.4 AUML and FIPA IPs

Half of the interaction protocols proposed by FIPA in AUML are illustrations of the use of a FIPA communicative act such as *request*, *request-when*, *query* and *propose* and how an agent responds to that communicative act. Even simple FIPA protocols, with only a few exchanged messages, contain a number of errors and imprecision. Some of these errors are inherent to the AUML notation and are present in all AUML protocols. A protocol has to be clear and complete because an agent may not have built-in experience about real life interactions or two or more agents may have

different intuitions, if any, about interactions. FIPA IPs and AUML tend to leave some actions and conditions implicit and dependent on the intuition of a developer. This section 5.4 points out errors recurrent in all the AUML specifications proposed by FIPA. Section 5.11 discusses the drawbacks of using AUML as a specification language, thereby justifying the need of a logic-based language like ANML.

Undefined conditions and guards

In most FIPA IPs, conditions and guards are not declared, initialised, set or reset anywhere in a protocol, for example the condition *agreed* in Figure 5.5. Conditions and guards, that are tested at a decision point, usually have to be related to previous messages. Undefined conditions wrongly regulate the message sequence. This problem is accentuated in realistic interaction protocols which are complex and involve more than a couple of messages being sent and one condition checked. They may contain iterations and nested protocols where guards (or conditions) must be set and reset. AUML overlooks these situations and adopts a simplistic view dependent on a developer's intuitive understanding of a condition and the message it seems to be related to. In ANML, the initialisation of conditions is explicitly defined and states are related to the processes that trigger them.

This problem is heightened when abbreviating and merging threads to occur along the same lifeline. Careless use of guards leads to confusion about which message triggered which condition. For example, in Figure 5.7 even though a participant responds with a *refuse* or *not-understood* message to a request from an initiator, the participant can still later send an *inform* that it has executed the initiator's request. Such errors occur in most of the FIPA IPs.

Any-time messages – timeouts and rejections

AUML does not support time management. In AUML, it is also difficult to represent a protocol that allows any participant to send a particular message at any time e.g. any agent can reject, send a failure or a timeout can occur at any point in an active negotiation. In order to show those messages that are possible at all times, each interaction thread in an AUML diagram must represent these messages at all the decision points. For example, an AUML diagram must show *all* agents checking that the interaction has not timed-out before sending *any* message and if not, then allow a

timeout to be sent. This leads to a cluttered diagram. On the other hand, it is easy to represent all-time valid actions like failure, reject and timeout messages in a statechart or in ANML. In ANML, only one rule is needed to allow a *timeout* to be executed from a parent state such as *open*.

Terminal states and processes, paths of actions

By abbreviating threads to one timeline, in AUML diagrams, it is not usually obvious what are the terminal actions and points. In some protocols, a *not-understood* or *failure* message terminate an interaction, while in other protocols proposals can still be sent after a *failure*. This is the case in Figure 5.8, where it is not clear where an interaction has ended and what is the result at the end. Similarly, possible sequences of messages are not clear when using one timeline.

Anonymous States

States are not explicitly shown in AUML. A message between agents ends in a vertical rectangle, which implicitly represent the point and state of the interaction. However such unnamed rectangles do not allow us to refer to the propositions that hold in a world or the beliefs of the participants at a particular point of an interaction. The state of an interaction before, during or after its execution cannot be referred to. States must be named because execution at an interaction thread usually depends on the current state of the interaction and on previous messages.

Raw values and deadlines

Raw values for time or deadlines or relative to the start of an interaction, are not prevented from being passed in AUML protocols. This produces errors and unattainable deadlines in case of iterative processes.

Imperfect communication layer

AUML assumes perfect communication in that messages are sent in the correct order and do not get lost. This is not realistic. There is no reasoning about the beliefs of a group of agents on the current state or point in the timeline. How does an agent send consecutive messages if it does not know whether its previous messages have been received and acted upon?

Executing decision points concurrently

Because states are not explicit and conditions and guards are used loosely in an AUML diagrams, decision points can be executed concurrently. This means that instead of sending a sequence of messages as in a conversation, all agents can send all their messages at one point in the communication. For example, in the protocol in Figure 5.4 an initiator can simultaneously send a call for proposal and an accept-proposal without waiting for responses from participants.

Roles and Instantiation

Roles are an interesting notion for modeling multi-agent systems. However, AUML and all FIPA IPs do not define the situation if more than one agent takes on an initiating role or if there are no participants. For example, what happens if there are no participants in Figure 5.4? AUML currently does not address the binding of roles or cardinalities to an agent's identity in case of deterministic decisions. It is hard to translate an AUML protocol to an executable since each agent would have *n* instances of the interaction for *n* participants. This issue is discussed in later sections of this chapter.

Figure 5.5 shows a *Request Interaction Protocol* (IP) proposed by FIPA, [FIPA 2001b], which consists of at most three messages being exchanged. We show that errors exist in even such a simple protocol. These errors originate from the specification of AUML and FIPA's use of AUML and are present in most of the FIPA IPs. The other FIPA protocols, analysed in sections 5.6 to 5.9, contain errors similar to those in Figure 5.5 and only additional errors in them are discussed.

5.5 FIPA *Request* Interaction Protocol

Figure 5.5 is a protocol in AUML for specifying sending and replying to a request. An initiator sends a *request* to a participant, who refuses, replies that it does not understand or agrees to satisfy the request and later reports on its success. This protocol can be represented as a path in ANML and a number of errors in it pointed out. These errors lead to an ambiguous and inconsistent interaction and wrong

beliefs. (Corrected versions of the protocol in both ANML and AUML are given later in this section.)

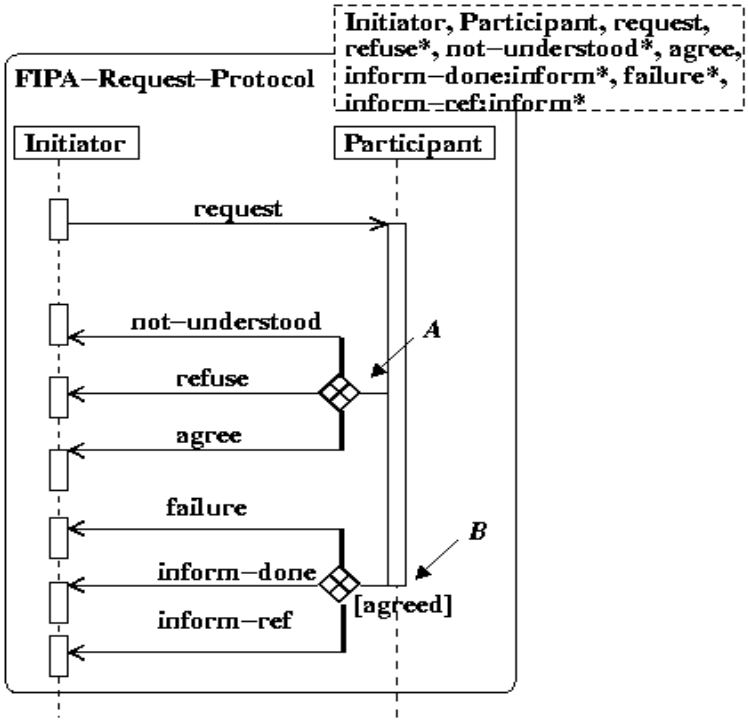


Figure 5.5 FIPA *Request* Interaction Protocol, [FIPA 2001b]

A literal translation of the *Request* protocol in Figure 5.5 along its timeline from AUML to an ANML path results in the following path of execution:

{*Initiator*:*initiator* , *Participant*:*participant*}. *FIPA-Request-Protocol* = *Initiator*. *request*; **(**one-of [*Participant*.*not-understood*, *Participant*. *refuse*, *Participant*.*agree*] ∧ (*agreed* → one-of ({*Participant*.*failure*, *Participant*.*inform-done*, *Participant*.*inform-ref*})) **)**

5.5.1 Errors in AUML *Request* Protocol

As remarked before, the condition *agreed* at point *B* in Figure 5.5 has no meaning for an agent and is not shown to be related to the previous *agree* message. The condition [*agreed*] is not defined, initialised, set or reset anywhere. Even though a participant sends *not-understood* or *refuse* after a *request*, it may still send *inform-done* or *failure* or *inform-ref* later. It is thus not obvious that *not-understood* and *refuse* messages lead to terminal states. In ANML, *agreed* can be treated as a state which is initialised to *false* and set to *true* when a participant sends an *agree* message.

In the AUML diagram, the request protocol is a path from a *request* to success or failure messages. The pre-conditions and post-conditions for an action are not well-defined and there are no constraints to stop an agent restarting a negotiation through a new request while in the middle of following the protocol. There is no facility for an agent to refer to a point in an interaction and to the effects of messages. All actions end up in anonymous points. What is the difference in the world between sending a *not-understood*, *refuse* or *agree* message since the state of the world or the effects of these actions are not given? How does an agent refer to the result of an interaction or differ between possible worlds as messages are sent? AUML protocols depend on the semantics of the FIPA ACL performatives.

According to the AUML notation, in Figure 5.5 there is no constraint against threads at *A* and *B* being executed concurrently. This leads to a number of contradictions with realistic interactions. A participant can send an *agree* or *not-understood* message while at the same time sending a *failure* or *inform-done*. Logically a participant should not be able to simultaneously send an agreement at point *A* and a failure at point *B*. In practice, [*agreed*] is initially *false* and depends on the decision at *A*. When messages at *A* and *B* are sent concurrently, even if a participant sends an *agree*, the condition [*agreed*] has no time to change from *false* and point *B* is not executed. Thus a participant does not execute a request even after agreeing to it and an initiator waits in vain for the result of an agreement. An agent may often need time to complete a task after agreeing to it and cannot execute *A* and *B* with success concurrently.

5.5.2 Corrections of the *Request* Protocol

The states of a negotiation are defined after each possible message by assuming that an (agent's) action triggers a corresponding state e.g. an *agree* action triggers an *agreed* state. The states after *request*, *refuse*, *inform-done*, *not-understood*, *agree*, *failure*, *inform-ref* actions are *requested*, *refused*, *informed-done*, *not-understood*, *agreed*, *failed* and *informed-ref* respectively. A protocol is ambiguous if any of the states are undefined at any point. It is incorrect if a state that becomes *true* is not as required by the semantics of the protocol, e.g. in an auction a bid triggers the *rejected* state. A complete protocol is sought where all states are well-defined at all instances by analysing the truth values of the states. The set of states is finite and bounded by

the request protocol since it does not contain any iteration. Before a negotiation, all the above states are initialised to *false*. To prevent an agent from restarting a negotiation whilst in the middle of one, the *interaction* state is introduced as the parent of all states; ¬*interaction* is *true* at the start of a negotiation and is the precondition for sending a request. Points *A* and *B* are also expressed as a sequential path.

Theory 5.6 is an axiomatisation in ANML of the request protocol by giving the relation between parent and sub-states and action-condition rules for state transitions. Worlds can be differentiated between by referring to the current state of a negotiation. The state and result of a negotiation can be identified at any point. Two more states *open* and *closed* are specified, as sub-states of *interaction*, to explicitly convey non-terminal and terminal states respectively. A *timeout* event is also allowed to occur at any time, leading to a *timedout* state. It can be proved that the final protocol, given by Theory 5.5, leads to well-defined states at all times. These corrections can be incorporated for a new AUML diagram and extended statecharts of the request protocol, given in Figure 5.6. Let *P* denote a participant, *I* denote an initiator and *X* as any agent.

interaction ↔ one-of ({*open*, *closed*})

open ↔ one-of ({*requested*, *agreed*})

closed ↔ one-of ({*not-understood*, *refused*, *failed*, *informed-done*, *informed-ref*, *timedout*})

¬*interaction* ↔ none-of ({*open*, *closed*})

¬*open* ↔ none-of ({*requested*, *agreed*})

¬*closed* ↔ none-of ({*not-understood*, *refused*, *failed*, *informed-done*, *informed-ref*, *timedout*})

¬*interaction* ↔ [*I*:*initiator.request*] *requested*(*I*)

requested(*I*) ↔ [*P*:*participant*. *not-understood*]*not-understood*(*P*) ∨ [*P*:*participant.refuse*] *refused*(*P*) ∨ [*P*:*participant.agree*] *agreed*(*P*)

agreed(*P*) ↔ [*P.failure*]*failed*(*P*) ∨ [*P.inform-done*] *inform-done*(*P*) ∨ [*P.inform-ref*] *informed-ref*(*P*)

$open \leftrightarrow [timeout]\ timedout$

Theory 5.6 Logical Theory of *Request* Protocol

Because the *Request* IP is such a simple protocol with at most 3 messages being sent out in a whole conversation, it is possible to represent corrected versions of the protocol in both AUML and statecharts notation, see Figure 5.6. The corrected AUML diagram explicitly contains which states hold and which states do not hold before and after both a decision point and sending a message. Showing states is not part of AUML, but this annotation of the AUML diagram is suggested in order to solve the problems in the original AUML request protocol. The annotation with states allows to define and set guards, to refer to what holds at a world during an interaction and not to execute all decision points concurrently. The protocol in Figure 5.6 shows timeouts as an optional message for each agent. In addition, to explicitly showing terminal points, the timelines in the AUML diagram must be split for each message received. The splitting of the timelines in Figure 5.6 is not shown so as not to clutter the diagram.

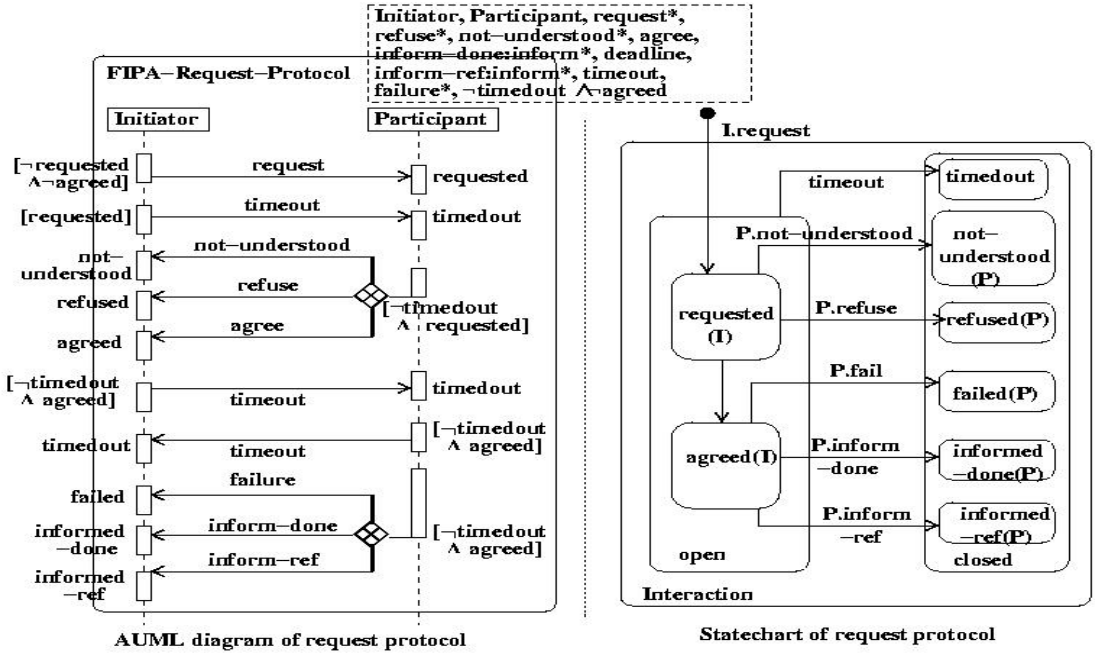


Figure 5.6 Suggested AUML and statechart diagram for *Request* IP

5.6 FIPA *Request-When* Interaction Protocol

The FIPA *Request-When* IP, [FIPA 2001c], illustrates the use of the FIPA ACL *request-when* communicative act. An initiator sends a *request-when* message to request that a participant do some action once a precondition becomes *true*. If the participant understands the request and does not refuse, it will *agree* and wait until the precondition occurs. Then, it will attempt to perform the action and notify the requester of success or failure. The participant can send a *refuse-2* message to the initiator, if it is no longer able to perform the action after agreeing.

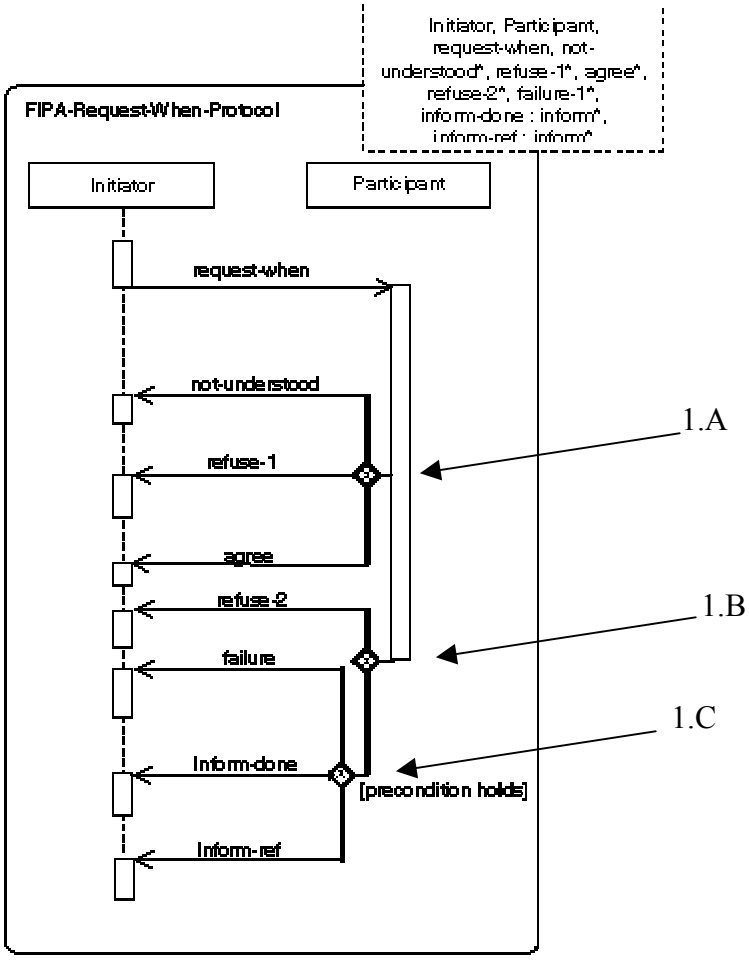


Figure 5.7 FIPA *Request-When* Interaction Protocol, [FIPA 2001c]

The translation of this protocol to a path in ANML is:

{*Initiator*:*initiator* , *Participant*:*participant* }.*FIPA-Requestwhen-Protocol* =
Initiator.request; one-of ({*Participant.not-understood*, *Participant.refuse-1*, *Participant.agree*}) ; (*precondition* → one-of ({*Participant.refuse-2, Participant.failure*, *Participant.inform-done*, *Participant.inform-ref*}))

5.6.1 Errors in *Request-When* IP

The condition [*precondition holds*]

The errors described in section 5.5.1 for the FIPA *request* IP apply here too since Figure 5.5 and Figure 5.7 specify nearly the same protocol. As for the condition [*agreed*] in the request protocol, here too the condition [*precondition holds*], at point *1.C*, has no syntax or semantic meaning. It is not declared, initialised and reset anywhere in the template specification. It is assumed that the semantics of some of the messages bear resemblance to some communicative acts in FIPA ACL. We assume that precondition is passed as a parameter in the *request-when* message from an initiator. The AUML diagram does not show the relation between the guard conditions and the *request-when* message. Any participant can send an *inform-done* irrespective of previous responses.

The condition *agreed*

Even a condition *agreed* is not shown at points *1.B* or *1.C*. Even if a Participant sends a *refuse* or *not-understood* message to a *request-when* message, the protocol allows it to progress along the lifeline and send *inform-done*, *inform-ref* or *failure* messages. The request-when protocol with a single lifeline, where terminal states are not clear, yields an undefined result at point *1.C*.

Deadlines

There are no deadlines for replying to a *request-when* or sending any other replies and no possibility of timeout events or cancelling whenever an agent wishes.

Redundancy at decision points

It is redundant to show points *1.B* and *1.C* as two different decision points. They can be merged as just one exclusive-or interaction threads with branches *refuse-2*, *failure*, *inform-done* and *inform-ref* and conditions along the branches.

5.6.2 Corrected *Request-When* IP in ANML

Theory 5.7 gives the *Request-When* Interaction Protocol in ANML. Rule (1) in Theory 5.7 allows initiator *I* to send a request-when message with condition *cond1*,

leading to the appropriate state change. Rule (2) enforces that the precondition *cond1* holds before a participant can respond.

interaction ↔ one-of ({*open*, *closed*})

open ↔ one-of ({*requested*, *agreed*})

closed ↔ one-of ({*not-understood*, *refused-1*, *refused-2*, *failed*, *informed-done*, *informed-ref*, *rejected*, *timedout*})

¬*interaction* ↔ none-of ({*open, closed*})

¬*open* ↔ none-of ({*requested*, *agreed*})

¬*closed* ↔ none-of ({*not-understood*, *refused-1*, *refused-2*, *failed*, *informed-done*, *informed-ref*, *rejected*, *timedout*})

¬*interaction*↔[*I*:*initiator*.*request-when*(*cond1*)](*requested*(*I*)∧*precon*(*cond1*)) (1)

(*requested*(*I*) ∧ *precon*(*cond1*)) ↔ *([P*:*participant*. *not-understood*]*not-understood*(*P*) ∨ [*P*:*participant*.*refuse-1*] *refused-1*(*P*) ∨ [*P*:*participant*.*agree*] (*agreed*(*P*) ∧ *precon*(*cond1*)) *)*

(*agreed*(*P*) ∧ *precon*(*cond1*)) ↔ ([*P*.*refuse-2*] *refused-2*(*P*) ∨ **(** *cond1* → [*P*.*failure*]*failed*(*P*) ∨ [*P*.*inform-done*]*inform-done*(*P*) ∨ [*P*.*inform-ref*]*informed-ref*(*P*)**)**) (2)

open ↔ [*reject*] *rejected* ∨ [*timeout*] *timedout*

Theory 5.7 *Request-When* Interaction Protocol in ANML

5.7 FIPA *Iterated Contract Net* Interaction Protocol

Having analysed two simple protocols in AUML, more realistic but relatively straightforward protocols are verified. It is found that their representation in AUML is incorrect and that in fact, AUML cannot completely express multi-agent protocols.

In the FIPA *contract net* protocol [Odell and al. 2000], a manager solicits proposals from other agents by issuing a call for proposals (*cfp*). Potential contractors receiving a *cfp* may reply with *propose*, *not-understood* or *refuse-1* before a deadline. After the

deadline, the manager sends an *accept-proposal* to selected agents and a *reject-proposal* to others. When a contractor has completed its task, it sends a completion message to the manager. The FIPA *iterated contract net* IP, [Fipa 2001e], is an extension of the contract net protocol to allow multi-round iterative bidding. After a first call for proposals, a manager may accept one or more of the bids, rejecting the others, or may repeat the process by issuing a revised *cfp* and rejecting all the proposals. The process terminates when the manager refuses all proposals and does not issue a new *cfp* or accepts one or more of the bids or the contractors all refuse to bid and the manager does not issue a new *cfp*.

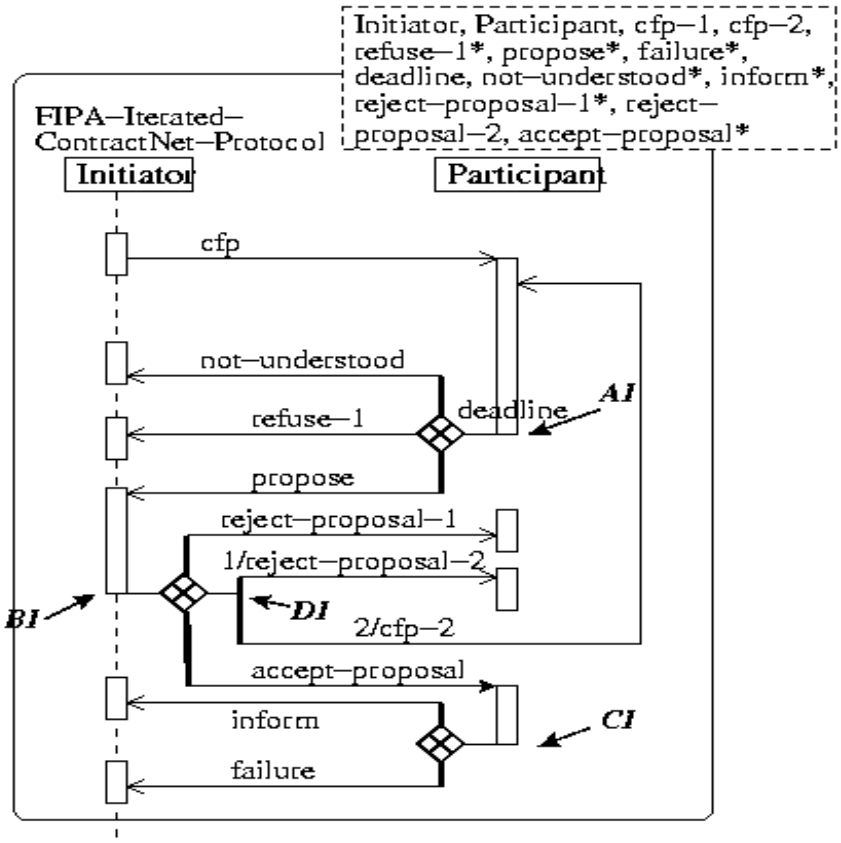


Figure 5.8 FIPA *Iterated Contract Net* IP, [FIPA 2001e, Odell and al. 2000]

The obvious error in Figure 5.8 is the participant's missing timeline. The protocol in Figure 5.8 is translated along its timelines into rules for a path in ANML. Let *P* denote a Participant and *I* an Initiator.

$$\{I{:}Initiator{:}initiator,\ P{:}Participant{:}participant\}.FIPA\text{-}IteratedContractNet\text{-}Protocol = I.cfp;\ \{I, P\}.\ contractNetProtocol$$

{*I*:*Initiator* , *P*:*Participant*}.*contractNetProtocol* = **(***deadline* ↔ one-of ({*P.not-understood*, *P.refuse-1*, *P.propose*}**)**) ; one-of({*I.reject-proposal-1*, {*I*, *P*}. *accepting*, **(***I.reject-proposal-2* ∧ (*I.cfp-2*;{*I* , *P*}.*contractNetProtocol* **)** })

{*I*:*Initiator* , *P*:*Participant* }.*accepting* = *I.accept-proposal*; one-of({*P.failure*, *P.inform*})

Theory 5.8 Path of *Iterated Contract Net* IP

5.7.1 Errors in *Iterated Contract Net* Protocol in AUML

Most of the errors mentioned in the previous sections also occur in the Iterated Contract Net protocol, Figure 5.8. Assuming that they are resolved, additional errors and rectifications are discussed here, while mentioning in brackets which rules in the ANML Theory 5.9 correct that error. Points *A1*, *B1*, *C1* and *D1* are referred to in Figure 5.8 in the following discussion of errors in the Iterated Contract Net IP. For example, a recurrent error is the lack of conditions or their incorrect setting at decision points. In Figure 5.8, even after a participant responds with a *not-understood* or *refuse-1* message at *A1* after a *cfp*, the interaction can continue and an initiator can send an *accept-proposal* at *B1*. The threads of interaction for both agents lying along one timeline require conditions at the decision points *A1*, *B1* and *C1*. Theory 5.9 gives a logical theory for this protocol after several revisions for ensuring that all states are well-defined and as expected.

Deadlines

In Theory 5.8 and Figure 5.8, a participant must wait for the deadline (at point *A1*) to occur in order to respond to a *cfp*. For example, if the deadline is set to 4 minutes then at exactly 4 minutes and 0 seconds, all agents must send their responses. This is not feasible and in practice, a participant is allowed to respond before a deadline. Deadlines that are given a ground value such as 9:00 am become unattainable when reiterating *cfp*s. Figure 5.8 also does not cover the case where a deadline passes without any participant sending any messages. Rules (26) and (20), in the corresponding ANML Theory 5.9, respectively specify a timeout event and the possible actions *before* a deadline.

Incompleteness

Point *B1* has no deadline or timeout for a manager to accept or reject proposals, neither does a participant have any deadline or timeout at *C1*. In fact, a number of events specified in the prose of the protocol are not portrayed in Figure 5.8. The case when all the contractors refuse to bid and the initiator does not issue a new *cfp* is not shown. Rules (10) and (22) capture this. The negotiation may also not terminate, ending up in infinite call for proposals, even if all agents send a refusal.

Scaling to multi-agent systems

The contract net protocol is a multi-agent interaction. Figure 5.8 shows an interaction between two roles, which is a concise way of portraying multi-agent interaction. However in a conversation, an agent may need to be deterministic and access information about a particular participant. The roles and cardinalities in an AUML diagram must be bound to the agent identities. For example, to show the winners of the contract net protocol in AUML, *n* roles for *n* winners are needed, leading to unreadable diagrams. Therefore to show a particular agent in AUML, a role has to be created for it. In the worst case scenario, this could escalate to showing as many roles as participants. The AUML notation needs to insert users in its specification.

The AUML notation does not show how an interaction in a multi-system depends on the responses of specific agents and their dynamic roles. How does an agent infer at point *B*, whether only one agent or a number of agents is sent an acceptance e.g. agents *X, Y, Z*, are to be sent acceptances? For example after *A1* in Figure 5.8, out of *n* agents, *m* agents send a proposal and *p* agents send a refusal. What is the relation between an agent among these *m* agents and one from the *p agents*? The identity of these *m* and (*n*-*m*) agents must be known so that each agent receives and sends the appropriate message. Only those agents whose proposals have been accepted should be able to send the results of executing a *cfp* at *C1*. There is a lack of guards and lack of information about which agent can perform which action. Figure 5.8 does not associate the identity of an agent to the messages being sent and it is not known which agents' proposals are accepted and which ones are rejected.

Figure 5.8 shows only two timelines. Illustrating a complete and reasonable interaction between more than 2 or 3 roles in AUML would be hard and would result

in an illegible diagram. In addition, AUML does not keep a parameter to record which agent sent which message to whom. What if participants wish to send messages between themselves as in a forum instead of just between an initiator and a participant? AUML cannot show voting-like protocols. When using roles, an agent *X* cannot base its decision on sending a message to agent *A* if it has sent or will send a message to agent *B*. It is hard to portray sequence of messages and decisions based on a sequence.

For *m*-*n* interactions in AUML, one would need to show all the timelines of all the agents in all their roles. This has to be specified beforehand and the number of agents fixed, preventing open interactions. Auctions are popular forms of negotiation allowing dynamic entries and thus cannot be realistically specified in AUML. Therefore AUML does not capture correctly multi-agent and dynamic negotiations with an agent changing roles dynamically.

Even if a group of (*n*+1) agents follows Figure 5.8 as a protocol, an initiator must keep *n* instances of the same negotiation for *n* participants, giving rise to complexity, concurrency and coordination problems. In an online auction, an auctioneer following an AUML protocol would need an auction instance for each (registered) participant, instead of only keeping track of the state of the auction with the bidders. If participants are allowed to know how the initiator responds to others, then each participant would need *n* instances of the interaction for each other participant. If an interaction allows communication between participants, then the *n* instances are related to each other leading to a complexity of m^n for *m* states.

In ANML the actions and states related to an agent are annotated with it. Finite set theory is used to manage groups of agents and parameterise actions and states with a group of agents. The sets of agents in each state are initialised to empty and an agent is added to an appropriate set according to the message exchange and the resulting state. For example, just after a *cfp*, *not-understood*({}) means that no agent has sent a *not-understood* message. When agents *ann* and *bob* send a *not-understood*, they are added to the set to give *not-understood*({*ann,bob*}).

Iterating call for proposals (*cfp*s)

After a participant responds with a *refuse-1* or a *not-understood* at *A1*, it is unclear in Figure 5.8 whether that participant is sent further *cfps* or it is written off for the rest of the interaction. In practice all agents should be sent revised *cfps*, since an agent which previously sent a refusal may later be able to execute a revised *cfp*. Figure 5.8 presupposes that only those participants that previously sent a proposal are eventually sent a revised *cfp*, as the interaction thread *D1* is a response to *propose*. In rules (7) and (9), a sub-state of *open* called *on-hold* is declared for those agents which sent a refusal or *not-understood* in response to a *cfp*. Rules (22) and (24) allow revised *cfps* to be sent to all agents and reinitialise the set of agents in the *on-hold* state.

Point *D1* is a concurrent interaction thread where an initiator simultaneously rejects the proposals of a number of agents and sends a new *cfp* to them. Confusion may arise when several messages are sent or received simultaneously. A participant may receive a revised *cfp* followed by a rejection because of delays in the communication channel. In this case, the participant's beliefs become inconsistent with the other agents. An initiator may also have difficulty in distinguishing between delayed proposals from previous *cfps*. *D1* should not be a concurrent thread and an initiator should first send a *reject-proposal-2* to some agents followed by a *cfp-2* to all agents. The conditions, states and sets of agents have to be re-initialised when reiterating a *cfp* as in rules (22) and (24) and the rules between states. It is unclear how to do this initialisation in AUML, even if guarding conditions were given. The first *cfp* is an entry point into a negotiation and must be distinguished from revised *cfps*. (rule 19).

Making Proposals

The AUML protocol supposes that as soon as an initiator receives a proposal, it cannot accept other messages and must perform the decisions at point *B1*. This behaviour is at odds with the English definition of the protocol where an initiator waits for a number of proposals before deciding on which ones to accept. Rule (8) solves this by making *proposed*(*Y*) be a sub-state of *cfped*(*X*).

In Figure 5.8, in order to proceed to point *B1*, at least one participant must send a proposal. An initiator cannot send a revised call for proposal unless he/she has received at least one proposal. This can lead to a deadlock because all participants

may refuse to bid and an initiator cannot revise its *cfp* from the protocol. The interaction does not terminate when no participants make a proposal. Rule (22) allows a timeout and sending a revised *cfp* in case of no proposals.

Terminal states

In AUML diagrams, abbreviating different threads of interaction onto a single timeline obscures the terminal actions and points for where a negotiation can end. It is not obvious in Figure 5.8 that an interaction ends when an initiator refuses all proposals and does not issue a new *cfp*. Contrary to Figure 5.5, in Figure 5.8 a negotiation is not closed if some agents send *not-understood* or *refuse-1* messages, since there can be revised *cfps*. In ANML, both *open* and *closed* states, their sub-states and the processes leading to them are obvious. States *failed*(*F*) and *informed*(*E*) may both be true at a point since some participants will send a successful completion while others a failure.

5.7.2 A logical theory of the *Iterated Contract Net* Protocol

Theory 5.9 is the ANML logical theory for an iterated contract net protocol after several iterations to ensure that the states of a negotiation are well-defined and as expected from valid actions. A multi-agent interaction is specified between an Initiator, *I*, and *n* other participants. Let *I*, *P* and *X* denote single agents and *Y*, *Z*, *A*, *B*, *C*, *D*, *E*, and *F* denote sets of agents. Rules (4) to (13) represent the relation between states and sub-states. Rules (14) to (18) ensure that messages are sent to the right agents or group of agents, so that an agent does not receive contradictory or unintended messages. Rules (19) to (26) are action-condition rules for state transitions.

$$interaction \leftrightarrow \text{one-of}(\{open, closed\}) \quad (4)$$

$$open \leftrightarrow \text{one-of}(\{cfped, pending\text{-}accomplishment\}) \quad (5)$$

$$pending\text{-}accomplishment(Y) \leftrightarrow (rejected\text{-}proposal\text{-}1(C) \vee proposal\text{-}accepted(Y\text{-}C)) \quad (6)$$

$$on\text{-}hold \rightarrow cfped \quad (7)$$

$$proposed \rightarrow cfped \quad (8)$$

$$on\text{-}hold \leftrightarrow not\text{-}understood \vee refused\text{-}1 \quad (9)$$

$closed \leftrightarrow$ one-of({($failed \vee informed$), $rejected$, $timedout$, $rejected\text{-}proposal\text{-}2$] (10)

$\neg interaction \leftrightarrow$ none-of ({$open$, $closed$}) (11)

$\neg open \leftrightarrow$ none-of ({$cfped$, $pending\text{-}accomplishment$ }) (12)

$\neg closed \leftrightarrow$ none-of({$failed$, $informed$, $rejected$, $timedout$, $rejected\text{-}proposal\text{-}2$}) (13)

($not\text{-}understood(A) \wedge refused\text{-}1(B) \wedge proposed(Y)) \rightarrow A \cap B \cap Y = \{\}$ (14)

($rejected\text{-}proposal\text{-}1(C) \wedge proposal\text{-}accepted(Z) \wedge proposed(Y)) \rightarrow$ (($Z = Y - C$) $\wedge\ C \cap Z = \{\}$) (15)

$failed(E) \wedge informed(F) \wedge proposal\text{-}accepted(Z) \rightarrow (F = Z - E)$ (16)

$failed(E) \wedge informed(F) \rightarrow (F \cap E = \{\})$ (17)

$rejected\text{-}proposal\text{-}2(C) \rightarrow proposed(C)$ (18)

$\neg interaction \leftrightarrow$ [I:$initiator.initial\text{-}cfp$] ($cfped(I) \wedge not\text{-}understood(\{\}) \wedge refused\text{-}1(\{\})$) (19)

($cfped(I) \wedge not\text{-}understood(A) \wedge refused\text{-}1(B) \wedge \neg proposed(Y)) \leftrightarrow$ one-of ({ [P:$participant$. $not\text{-}understood$]$not\text{-}understood(A \cup \{P\})$, [P:$participant$. $refuse\text{-}1$] $refused\text{-}1(B \cup \{P\})$, [$P$:$participant.propose$] $proposed(\{P\})$}) (20)

($cfped(I) \wedge not\text{-}understood(A) \wedge refused\text{-}1(B) \wedge proposed(Y)) \leftrightarrow$ one-of({ [P:$participant$. $not\text{-}understood$]$not\text{-}understood(A \cup \{P\})$, [$P$:$participant$. $refuse\text{-}1$] $refused\text{-}1(B \cup \{P\})$, [$P$:$participant.propose$]$proposed(Y \cup \{P\})$]}) (21)

($on\text{-}hold \wedge \neg proposed(Y) \wedge cfped(I)) \leftrightarrow$ [$timeout$; $I.cfp$] ($cfped(I) \wedge not\text{-}understood(\{\}) \wedge refused\text{-}1(\{\})$) (22)

$proposed(Y) \leftrightarrow$ ([I:$initiator.reject\text{-}proposal\text{-}1$] $rejected\text{-}proposal\text{-}1(C) \wedge$ [I:$initiator.accept\text{-}proposal$] $proposal\text{-}accepted(Z) \wedge Z = Y\text{-}C) \vee$ ([I:$initiator.reject\text{-}proposal\text{-}2$]$rejected\text{-}proposal\text{-}2(Y)$ (23)

$rejected\text{-}proposal\text{-}2(Y) \leftrightarrow$ [I:$initiator.cfp$] ($cfped(I) \wedge not\text{-}understood(\{\}) \wedge refused\text{-}1(\{\})$) (24)

$$proposal\text{-}accepted(Z) \leftrightarrow [P{:}participant.inform]informed(E) \vee [P{:}initiator.failure]$$
$$failed(F) \wedge E = Z\text{-}F \quad (25)$$
$$open \rightarrow [reject]\ rejected \vee [timeout]\ timedout \quad (26)$$

Theory 5.9 *Iterated Contract Net* Protocol in ANML

5.8 FIPA *English Auction* Interaction Protocol

Figure 5.9 shows an *English Auction* Interaction Protocol (IP), [Odell and al 2001], in AUML. An auctioneer initially proposes a price for a posted item and then raises the price when receiving a bid. The auctioneer waits for a proposal to his *cfp* and as soon as a buyer indicates that it will accept the price, the auctioneer issues a new call for bids with an incremented price. The auction continues until no buyers are prepared to pay the proposed price. The good is sold to the last buyer who made a proposal or it is not sold at all depending on the strategy of the auctioneer. This is different from the usual English auction, in which a bidder raises the price. Here the auctioneer is given more power and decides on how much to increase the price and can decide to withdraw from the auction if its reservation price is not met.

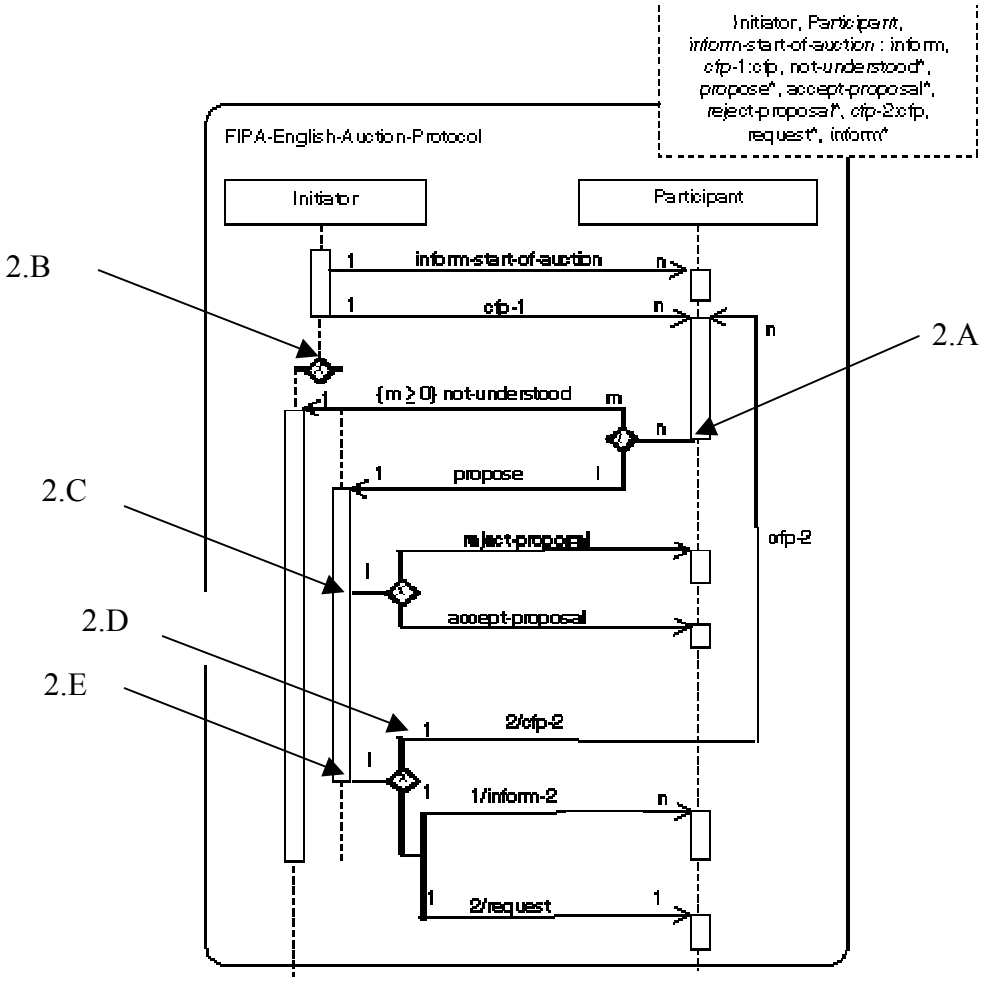


Figure 5.9 *English Auction* IP, [FIPA 2001h], [Odell and al 2001]

The translation of the AUML *English Auction* protocol into an ANML path is:

{*I*:*Initiator* , *P*:*Participant*}.*FIPA-EnglishAuction-Protocol* = *I.inform-start-of-auction*; *I.cfp*; {*I* , *P*}.*English-auction*

{*I*:*Initiator* , *P*:*Participant*}.*English-auction* = one-of ([*P.not-understood*, (*P.propose*; {*I* , *P*}.*after-propose*)})

{*I* :*Initiator* , *P*:*Participant* }.*after-propose* = one-of ([*I.reject-proposal*, *I .accept-proposal*}); one-of ({(*I.cfp-2*; {*I* , *P*}.*English-auction*), (*I.inform* ∧ *I.request*)})

5.8.1 Errors in *English Auction* IP in AUML

Binding Roles

At the end of the protocol, an *inform* message is sent to *n* agents and a *request* to *1* agent. One can guess that the *inform* messages are to let *n* agents know that their proposal has been rejected and the *request* message is to tell 1 agent that its proposal

is accepted and to request completion of the proposal. However, the AUML protocol does not embed this information. The protocol could be interpreted as sending an *inform* to *n* agents to tell them that their proposals have been accepted. The AUML protocol does not specify to which participants, an initiator should send *inform* and *request* communicative acts. There is no relation in the AUML diagram between points *2.C* and *2.E*, i.e. between proposers and those who are sent inform and request messages.

As in the Iterated Contract Net IP, there is no concept of an agent's identity in the *n* (or *m*) agents in Figure 5.9. If *m* agents send a not-understood message and the initiator send an accept to *n* agents, then where does the protocol ensure that any of the *m* agents are not part of the *n* agents? The problem is how are the numbers *m* and *n* related to the progress of an interaction and the agents sending a particular message.

Error in mathematics in the use of numbers *m* and *n*

The protocol is not open and allows only *n* participants being sent an *inform-start-of-auction* and any message. Even then, the numbers *m* and *n* are used carelessly throughout the AUML Figure 5.9, leading to incorrect messages being sent.

- At point 2.E, *1* agent (the winner) is sent a request and *n* (all participants) agents are sent an *inform-2*. The winner thus gets both an *inform-2* of rejection and a *request* to satisfy its proposal.
- Given that AUML represents interactions through roles, then it is incorrect to show *m* and *n* cardinalities at a participant's role at point *2.A*. It could be interpreted as a participant sending *m* not-understood messages, instead of one. For this, the AUML notation must be changed to show that it is an initiator who receives *m not-understood* messages.
- The AUML protocol specifies that *n* agents are present initially, *m* out of *n* agents reply with a *not-understood* to an initial *cfp* and just *1* agent proposes. Therefore, at the end at point *2.E*, the initiator should send an *inform* message of rejection to at most (*n-m-1*) agents instead of to every *n* agents. In the AUML protocol, those who sent a *not-understood* or did not reply to a *cfp* are informed that their proposals are rejected.

Storing the last highest bidder before a revised *cfp*

In the AUML Figure 5.9, an auctioneer rejects agent's *A*'s proposal and sends a new *cfp*, then it accepts *A*'s old proposal later when its revised *cfp* has failed. Agent *A* receives a rejection then an acceptance and would be confused as to the auctioneer's intentions. Similarly after an agent *B* receives an *accept-proposal*, the auctioneer may receive another better proposal from another agent after a revised *cfp*. Agent *B* then receives then an *inform* of rejection after its proposal had been accepted.

Figure 5.9 specifies that after a proposal, the auctioneer there and then decides whether to accept or reject the proposal, defeating the purpose of a revised *cfp*. Points *2.C* and *2.E* are wrongly ordered with respect to each other.

A fragment of what the protocol should allow according to the requirements is: if agent *X* proposes, a *cfp* is immediately sent. Then if agent *Y* proposes, a reject is sent to *X* and another *cfp* is sent and the whole process is repeated. An agent's proposal is accepted if there are not further proposals after a *cfp*. The path in ANML between initiator *I* and participants *X* and *Y* is:

(*X.propose*; *I.send-cfp*; *Y.propose*; (*I.reject-to-X* ; *I.send-cfp*))*

This path is repeated until there are no more proposals to a revised *cfp* and then the last proposal before the revised *cfp* is accepted.

Therefore, the auctioneer needs to first store the last proposal it obtained (from say *X*) before sending a revised *cfp*, then store *Y* as the new proposer and finally send a rejection to *X*. AUML notation does not allow storing the history of an interaction and therefore there is no way to portray storing previous proposals and to correctly show even a simple protocol like the English auction.

What happens to those agents who did not respond?

The AUML protocol assumes that all agents reply to a *cfp*, implying that all agents who do not send an *not-understood* message should send a proposal. This is enforced by the cardinality *n* at point *2.A*. Given that the number of agents not responding is unknown, then the number of *informs* to be sent at point *2.E* cannot be specified, (at most *n-m-1*).

Agents sending a *not-understood* are not sent any revised *cfp-2*

Even if it is the initiator's mistake in the original *cfp*, those who sent a *not-understood* message are written off the interaction forever. This is given by the *not-understood* timeline tailing off to the end of the diagram. There is a contradiction when a revised *cfp* at point *2.D* are sent to all *n* agents.

Concurrent threads *2.C* and *2.E*

According to Figure 5.9, points *2.C* and *2.E* can be performed concurrently when they must be sequential. This allows an initiator to *reject* proposals and perform *cfps* and *informs* all simultaneously.

Deadlines

In real life and online auctions, there are deadlines for bidding after a *cfp* and for accepting a proposal. The AUML Figure 5.9 does not show any deadlines at points *2.A*, *2.C* and *2.E*.

Incompleteness

The AUML diagram does not give all termination actions. It does not show that the auction terminates if no buyers send a proposal and the auctioneer no longer sends *cfp*s or if the auctioneer rejects all proposals and does not sell the item.

No conditions

Conditions before performing the different threads of interaction are not given e.g. there has to be a proposal before accepting or rejecting a proposal or before performing a second *cfp*, an *inform* or a *request*. The protocol does not show that an auctioneer sends a new *cfp* as soon as there is a bid from a participant.

Non-terminating if no proposals

An auctioneer must receive a proposal before he can do any *accept* or *reject* of previous proposals or send a revised *cfp*. If no agents propose, the interaction hangs and does not terminate.

Redundancy in AUML protocol

There is no need to perform the *inform* and *request* messages at point *2.E* if an auctioneer has already sent an *accept-proposal* or a *reject-proposal* before to a participant. Instead, these two messages can be taken as informing the participants about the end of the auction.

5.8.2 Corrected *English Auction* IP in ANML

The ANML theory of the FIPA English auction is given below and is closer to its requirements than Figure 5.9.

$interaction \leftrightarrow$ one-of ($\{open, closed\}$)

$open \leftrightarrow$ one-of ($\{informed\text{-}started, first\text{-}cfped, just\text{-}proposed, accepted\text{-}proposal\}$)

$not\text{-}understood \rightarrow open$

$closed \leftrightarrow$ one-of ($\{ (informed \vee requested), withdrawn, timedout, rejected\}$)

$just\text{-}proposed(X) \leftrightarrow$ one-of ($\{proposed(X) , iterated\text{-}cfp(I)\}$)

$rejected\text{-}proposal \rightarrow proposed$

$\neg interaction \leftrightarrow$ none-of ($\{open, closed\}$)

$\neg open \leftrightarrow$ none-of ($\{informed\text{-}started, first\text{-}cfped, just\text{-}proposed, accepted\text{-}proposal\}$)

$\neg closed \leftrightarrow$ none-of ($\{ informed, requested, withdrawn, timedout\}$)

$\neg just\text{-}proposed(X) \leftrightarrow$ none-of ($\{proposed(X), iterated\text{-}cfp(I)\}$)

$proposed(Y) \wedge not\text{-}understood(C) \rightarrow (C \cap Y = \{\})$

$\neg interaction \leftrightarrow [I{:}initiator.inform\text{-}start\text{-}of\text{-}auction]\ informed\text{-}started(I)$

$informed\text{-}started(I) \leftrightarrow [I.cfp]\ (first\text{-}cfped(I) \wedge not\text{-}understood(\{\}))$

$(first\text{-}cfped(I) \wedge not\text{-}understood(A)) \leftrightarrow [P{:}participant.\ not\text{-}understood]\ not\text{-}understood(A \cup \{P\}) \vee [P{:}participant.propose]\ proposed(P)] \vee [timeout; I.cfp]\ (first\text{-}cfped(I) \wedge not\text{-}understood(\{\}))$

$proposed(P{:}participant) \leftrightarrow [I{:}initiator.cfp]\ (iterated\text{-}cfped(\{I\ ,\ P\}) \wedge not\text{-}understood(\{\}))$

$iterated\text{-}cfped(\{I{:}initiator\ ,\ P{:}participant\ \}) \wedge not\text{-}understood(A) \leftrightarrow$ one-of ({

$[X.\ not\text{-}understood]not\text{-}understood(A \cup \{X\})$,

$[G{:}\ participant.propose;\ (\{I,\ P\}.\ reject\text{-}proposal)]\ (proposed(G) \wedge rejected\text{-}proposal(P))$,

$[timeout;\ (\{I,\ P\}.\ accept\text{-}proposal)]accepted\text{-}proposal(P)$,

$[timeout;\ (\{I,\ P\}.\ reject\text{-}proposal)]\ rejected\ \}$)

$\{I{:}initiator\ ,\ P{:}participant\ \}.\ accept\text{-}proposal = I.accept\text{-}proposal$

$\{I\ {:}initiator\ ,\ P{:}participant\ \}.\ reject\text{-}proposal = I.reject\text{-}proposal$

$accepted\text{-}proposal(P{:}participant) \leftrightarrow [I{:}initiator.request]\ requested(P) \wedge [I{:}initiator.inform]informed(X)$

$open \rightarrow [I{:}initiator.withdraw]\ withdrawn$

$open \wedge \neg(first\text{-}cfped(I)\ \vee iterated\text{-}cfped(\{I\ ,P\})) \rightarrow [timeout]\ timedout$

5.8.3 *Dutch Auction* IP

In the FIPA *Dutch Auction* Interaction Protocol (IP), [FIPA 2001i], an auctioneer starts the bidding at a price much higher than the expected market value, then progressively reduces the price until one of the buyers accepts the price. If the price reaches the reserve price without any bids, then the auction terminates. The protocol allows for a bid to be rejected in the case of multiple, competing and simultaneous bids. Many of the errors in the AUML Dutch auction diagram, [FIPA 2001i], resemble those of the English auction AUML Figure 5.9 e.g. cardinalities *m* and *n* denoting a number of agents in both protocols are not defined and are wrongly assigned to the messages being sent. Deadlines and conditions for sending bids and accepting them are undefined. There are ambiguities with concurrent actions and exclusive-or actions. Further errors applying only to the FIPA Dutch Auction protocol are:

- Lack of declaration and setting for shown conditions e.g. *no-bids*

- It is possible for an auctioneer to accept a proposal then send another call for proposal when in practice a Dutch auction is stopped once the auctioneer accepts a proposal.
- Again termination conditions are absent or unclear e.g. when there are no bidders or the auctioneer rejects all proposals.
- There are issues about concurrency such as an auctioneer can simultaneously accept a proposal and send a *cfp*.

5.9 FIPA *Recruiting Interaction* Protocol

In the FIPA brokering protocol, [FIPA 2001f], an initiator requests a broker to find one or more agents who can satisfy a query. The broker then determines a set of appropriate agents to which to forward the query, sends the query to those agents and relays their answers back to the original requestor. The FIPA Recruiting Interaction Protocol (IP), [FIPA 2001g], is similar to the brokering protocol except that results of a query from selected server agents may be sent directly to the original requestor or some designated receiver agent instead of via the broker. This section discusses the FIPA recruiting protocol and most of the remarks apply to its brokering protocol too.

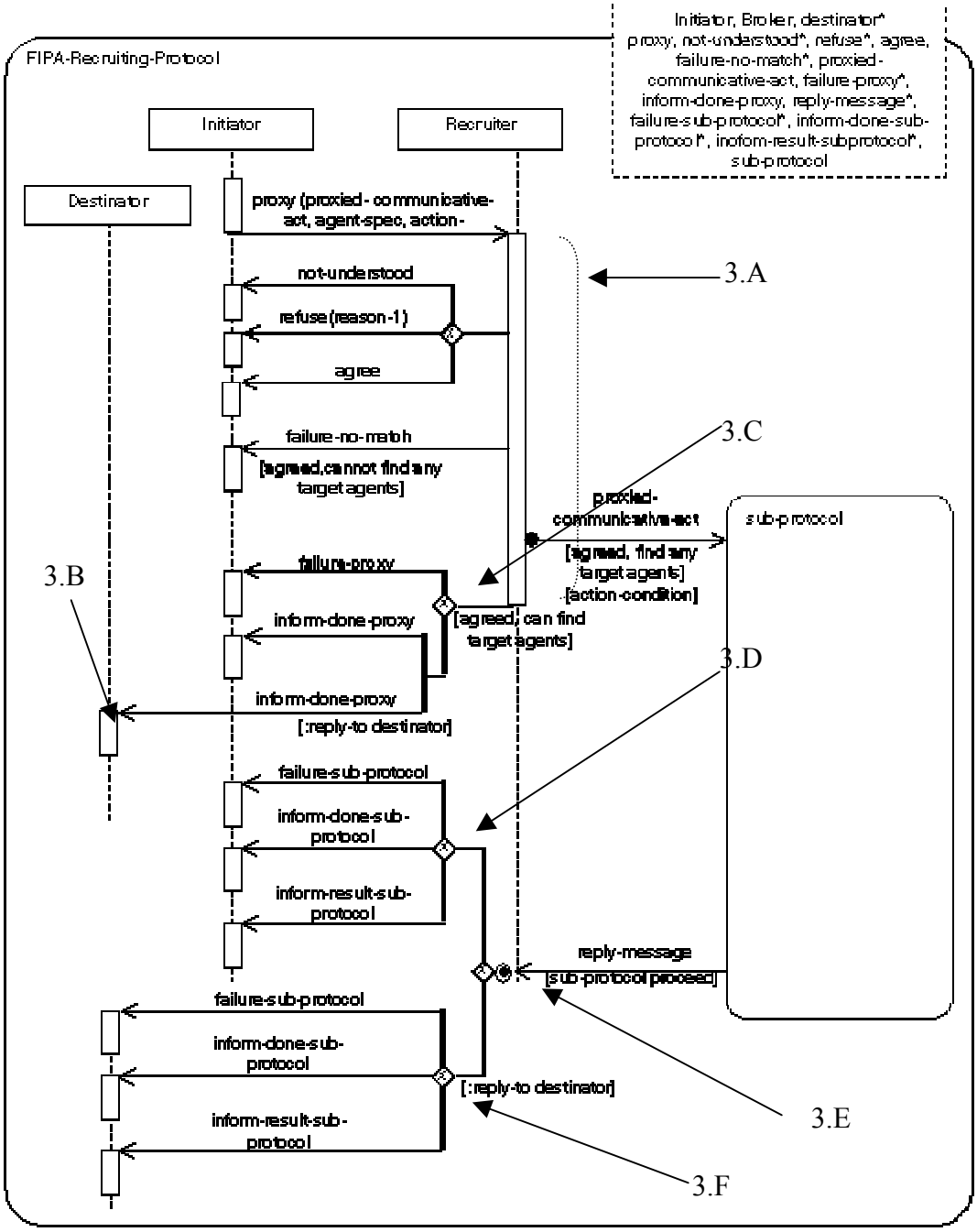


Figure 5.10 FIPA *Recruiting Interaction* Protocol, [FIPA 2001g]

ANML initial theory by literal translation of Figure 5.10:

{*I*:*Initiator* , *R*:*Recruiter* , *D*:*Destinator* }.*FIPA-Recruiting-Protocol* = *I.proxy*(*proxied-communicative-act, agent-spec, action*); one-of ({*R.not-understood*, *R.refuse*(*reason-1*), *R.agree*}) ; ((*agreed* ∧ *cannot find target agents*) → *R.failure-no-match*) ; ((*agreed* ∧ *can find target agents*)→ *launch-sub-protocol* ∧ one-of({*R.failure-proxy*, *R.inform-done-proxy*[2] }) ; *result-out-of-subprotocol*

result-out-of-subprotocol = one-of ({ one-of ({ *failure-sub-protocol*, *inform-done-protocol*, *inform-result-subprotocol*}), (*reply-to destinator* → one-of [*failure-sub-protocol*(*D*), *inform-done-protocol*(*D*), *inform-result-subprotocol*(*D*)})]

Theory 5.10 An Initial Theory for the *Recruitment* protocol in ANML

5.9.1 Some Errors in AUML *Recruitment IP*

Undefined terms and conditions

Terms and conditions are used throughout the protocol without being defined anywhere. For example, in the message *proxy*(*proxied-communicative-act, agent-spec*, *action*), only the parameter *proxied-communicative-act* is declared in the template. Similarly, undefined conditions are [*agreed, can/not find any target agents*], [:*reply-to destinator*], [*sub-protocol proceed*], [*agreed, find any target agents*], [*action-condition*]. What are the conditions for launching a sub-protocol? The AUML protocol does not relate these conditions to previous messages.

Concurrent Actions

In Figure 5.10, all the actions along the timeline at *3.A* can be executed concurrently. According to the AUML protocol, after receiving a request from an Initiator, a recruiter may send an agreement, a failure of finding a server agent or inform-done and launch a sub-protocol all simultaneously. In practice, these messages are sequential, allowing guards to be set before being tested.

Nesting of Protocols, missing roles and inadequate information

What is the empty rectangle *sub-protocol* and how does a recruiter know what to perform when launching *sub-protocol*? The role of a server is missing from Figure 5.10.

It is unclear whether the AUML protocol in Figure 5.10 is meant to be a shared protocol or the protocol known by only the *recruiter*. In either case, there is not enough information for the participants about allowed actions. It is not enough for a *recruiter* to be given a link to an unspecified sub-protocol as in Figure 5.10.

On the other hand, an *initiator* does not need to know about the *recruiter* internally launching a sub-protocol or sending a message to a *destinator*. It is not in the concern of an *initiator* how the *recruiter* gains results to a query. Therefore Figure 5.10 does not show enough information for a *recruiter* or a *server*, but shows too much information for the *initiator* and *destinator*. If Figure 5.10 is a shared protocol, then all interactions between the four roles – *server*, *initiator*, *recruiter* and *destinator* have to be completely defined.

ANML adopts the view that a shared protocol is known by all the agents, and each one of them may adapt the protocol to their own purpose.

Point *3.E* executed in all cases

Because the conditions at point *3.E* are incomplete, the actions at this point are always executed. Thus, a *server* informs an *initiator* or *destinator* of the results of a query even if the *recruiter* previously replied that it did not understand a message, that it failed to find an appropriate *server* or that the *server* failed.

Cannot send whether-succeeded and results together

Point *3.D* and *3.F* are exclusive-or decision points. A *server* performs either *failure-sub-protocol*, *inform-done-sub-protocol* or *inform-result-sub-protocol*, but not all three. So a *server* can tell an *initiator* or *destinator* that its request has successfully been achieved but cannot inform it of the results.

The condition *inform-result-sub-protocol* is an example of the confusion arising when using textual conditions. Does a *server* inform an *initiator* the results of its request or the results of launching a sub-protocol?

When there is no destinator

The message *Inform-done-proxy* at pint *3.B* is sent both to the *initiator* and to the *destinator*. This leaves a dangling *inform-done-proxy* at point *3.B* when there is no *destinator*.

5.9.2 ANML Theory for Recruiting Protocol

The logical theory for a recruiting protocol in ANML aims to solve the above errors between agents *I*:*initiator*, *R*:*recruiter*, *D*:*destinator* and *Servers*: *server*.

interaction ↔ one-of [*open*, *closed*]

open ↔ one-of [*proxied*, *agreed*, *informed-done*, *replied-to-who*, *informed-done-destinator*, *waiting*]

closed ↔ one-of [*not-understood, refused, failed-no-match, failed-server-communication, failed-query-results, request-succeeded*]

¬*interaction* ↔ none-of [*open*, *closed*]

¬*open* ↔ none-of [*proxied*, *agreed*, *informed-done*, *replied-to-who*, ¬*informed-done-destinator*, *waiting*]

¬*closed* ↔ none-of [*not-understood, refused, failed-no-match, failed-server-communication, failed-query-results, request-succeeded*]

¬*interaction* ↔ [*I*:*initiator*.*proxy*] *proxied*(*I*)

proxied(*I*) ↔ [*R*:*recruiter*.*not-understood*] *not-understood* ∨ [*R*:*recruiter*.*refuse*] *refused* ∨ [*R*:*recruiter*.*agree*] *agreed*(*R*)

agreed(*R*) ↔ [*R*.*failure-no-match*] *failed-no-match* ∨ [*R*.*failure-communication*] *failed-server-communication* ∨ [*R*.*inform-done-proxy*]*server-sub-contracted*(*R*)

server-sub-contracted(*R*) ↔ [*I*:*initiator*.*reply-to-who*(*D*:*destinator*)] *replied-to-who*({*I*, *D*})

replied-to-who({*I*:*initiator* , *D*:*destinator* }) ↔ **(** ¬(*I*=*D*) → [*R*:*recruiter*.*inform-done-destinator*] *server-sub-contracted-Destinator*({*R*, *D*}**)) ∨ ((***I*=*D*) → [*I*.*wait*]*waiting*(*I*)**)**

server-sub-contracted-Destinator({*R*:*recruiter*, *D*:*destinator* }) ↔
[*D.wait*]*waiting*(*D*)

waiting(*D*) ↔ [*S*: *server.server-failure-query(D)*] *failed-query-results* ∨ [*S*: *server.server-done-query(D)*; *S*: *server.server-results-query(D)*] *request-succeeded*

Theory 5.11 Theory for *Recruiting* Protocol

5.10 Translating Bilateral negotiation from ANML to AUML

The above sections show representing AUML diagrams in ANML after verifying them. The reverse process is now carried out by converting the ANML theory of the bilateral protocol, Theory 5.5, into an AUML interaction diagram. Let the bilateral protocol consist of two roles, *Agent1* and *Agent2*. Already the distinction between these two roles is not clear. Figure 5.11 shows the bilateral protocol as an AUML diagram.

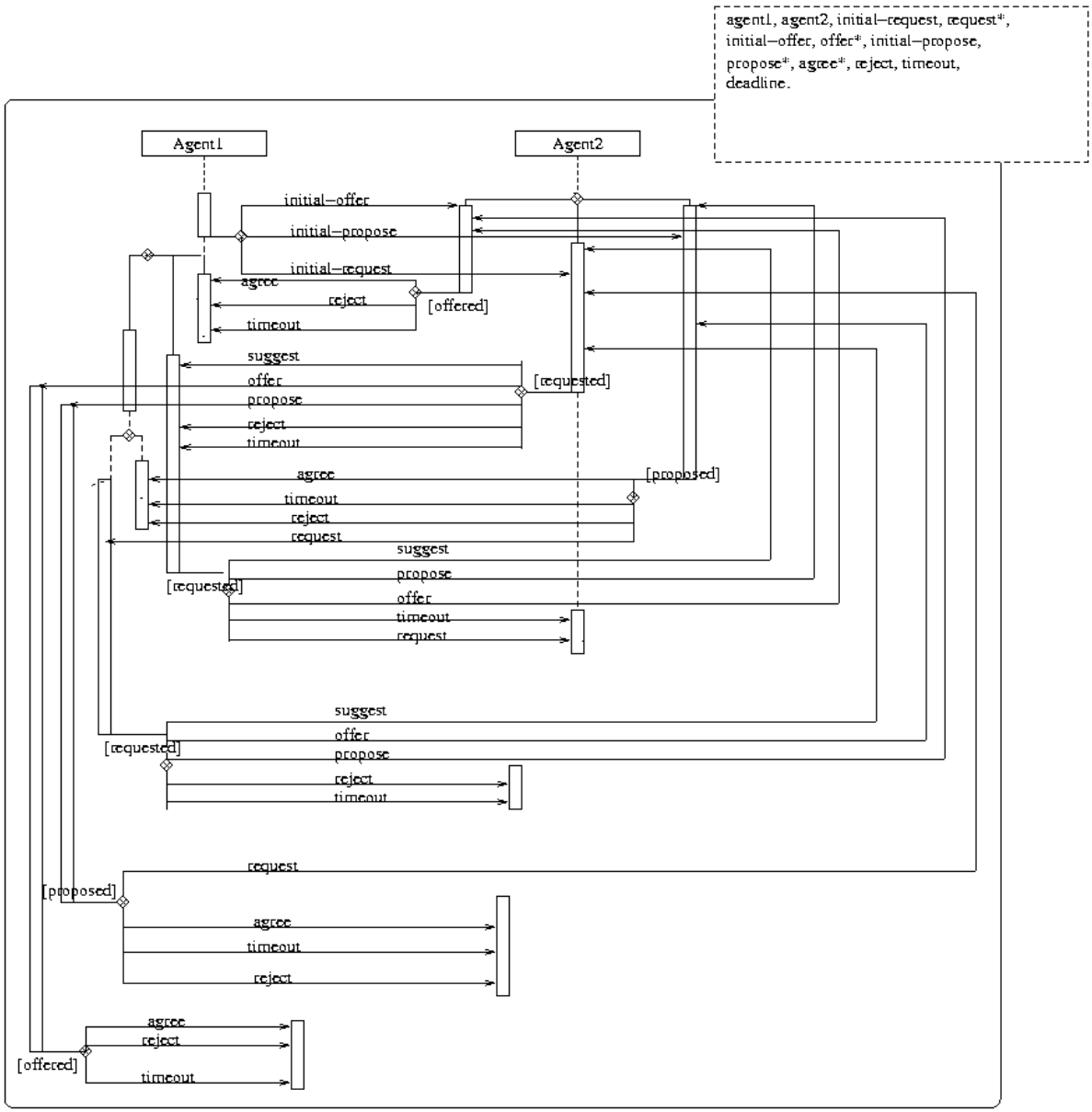


Figure 5.11 Bilateral Protocol in AUML

As can be seen, a representation in AUML of a simple theory like the bilateral protocol produces a complicated diagram, especially with timeout events and reject actions.

In the AUML diagrams in the above sections, it is rare to find decision points with more than one non-terminal message. All messages, apart from one thread, end the interaction e.g. *not-understood*, *fail* or *refuse*. This is why the threads of interaction could be abbreviated to a single timeline in AUML. However, in the bilateral protocol, there are several non-terminating messages that can be sent at a decision

point. Therefore threads of interaction at the decision points cannot be abbreviated to a single timeline as the consequent events differ substantially depending on the messages being sent.

Dynamic roles in the bilateral protocol are another reason for the complexity of Figure 5.11. The roles of the agents are not fixed since they are not identified as being an initiator or participant and any of the two involved agents can perform an action depending on the last state. In AUML, this introduces inevitable redundancy on each agent's timeline since a decision point is needed for each condition *requested*, *proposed* or *offered* for each role.

5.11 What is wrong with AUML?

Most of the FIPA proposals for interaction protocols illustrate the use of a FIPA communicative act. There are no protocols involving bargaining or dialogues, which are found difficult to represent correctly in AUML.

Undefined conditions

Because an AUML diagram does not show the relation between states and processes, the conditions at decision points are informal, are not declared and cannot be related to previous messages.

No binding of roles or cardinalities

Roles are useful for portraying multi-agent interaction, but are not sufficient on their own. An agent may need to be deterministic and access information about a particular participant. It needs to know which agents have been sent which messages or to which agents should a response be sent. Variables such as *m* and *n* are used extensively but wrongly in AUML.

We must be able to bind the roles and cardinalities in an AUML diagram to the agent identities. For example, to show the winners of the contract net protocol, *n* roles are needed for *n* winners. Therefore a role would have to be created for each agent to be identified. In the worst case scenario, this could escalate to showing as many roles as participants. The AUML notation needs to insert users in its specification. AUML

cannot show voting-like protocols and cannot illustrate an agent sending a message to specific participants, e.g. accepting only 2 or 3 proposals from and to known agents.

Cardinalities denoted at the wrong role

Given that AUML represents interactions through roles, cardinalities are specified at the wrong role. There is confusion about whether a participant sends *m* messages or *m* participants send one message. The AUML notation must distinguish between these two cases and instead show an initiator receiving *m* messages.

Cannot represent *n-m* interactions

Illustrating a complete and reasonable interaction in AUML between more than 2 or 3 agents would be hard and would result in an illegible diagram. There is currently no legible way to show participants sending messages between themselves as in a forum instead of just between an initiator and a participant. For a multilateral IP in AUML involving more than two roles, one would need to show all the timelines of all the agents.

Dynamic Interactions

Showing dynamic interactions like the bilateral protocol results in an illegible diagram because AUML deals only with roles. In AUML, all the decision points possible at each state will have to be shown on each agent's timelines, i.e. decision points at *requested*, *offered, proposed* will have to be shown *n* times for *n* negotiating agents.

Open Interactions

It is not possible to specify open interactions in AUML where agents can join dynamically in a negotiation. This is because cardinalities are used at each interaction thread as in the protocol for the English auction, Figure 5.9. Cardinalities are also inappropriate when there are agents who do not respond to a message as in an auction.

Instances of AUML protocols

It is currently not possible to translate an AUML protocol into an executable. In addition to needing to insert users, each agent needs *n* instances of the protocol for *n* participants. This leads to complexity and coordination problems.

No time management

AUML cannot represent time-dependent actions or ubiquitous messages. For example, in order to represent a ubiquitous message like *reject* or an event such as *timeout*, an AUML diagram has to show these as possible at each thread of interaction. In addition an AUML protocol has to show testing that the interaction is still open whenever a message can be sent.

No notion of history

An agent *X* cannot base its decision on sending a message to agent *A* if it has sent or will send a message to agent *B*. It is hard to portray sequence of messages and decisions based on a sequence or conditional behaviour. As found in the English auction, AUML diagrams cannot store information about previous messages and who sent them. AUML loses also the advantage of providing planning capabilities that would otherwise be possible in a formal methodology.

Too specific for standardisation

AUML and FIPA Interaction Protocols are too specific for standardisation. They depend on agents adopting roles and sending FIPA-like messages. Instead, they may act as a repository of templates for protocols, which are instantiated for a negotiation. The agents still must use state and activity diagrams for their reasoning. "AUML is an agent-centric view on specifying protocols which emphases an agent and its roles first and the interaction second", [Bauer and al. 2001]. Are constraints embedded in the roles? This causes problems in interactions where the roles of an agent do not matter and they can take on each other's roles.

OCL – Object Constraint Language

There has been mention of using OCL as a constraint language, but OCL is not formal, [Richters and Gogolla 1998], and addresses just pre- and post-conditions in object oriented UML.

More than one non-terminal message as a decision point

In the analysed AUML diagrams, it is rare to find decision points with more than one non-terminal message. All messages, apart from one thread, end the interaction e.g. *not-understood*, *fail* or *refuse*. This is why the threads of interaction could be

abbreviated to a single timeline. However, in realistic protocols, there are several non-terminating messages that can be sent at a decision point. Then, threads of interaction cannot be abbreviated to a single timeline and for each message a new timeline for the same role is needed.

Deadlines

In addition to not addressing cancellation, timeout, inputs and crash events happening at any point in an interaction, AUML does not show how to deal with unexpected messages or delayed messages through communication problems. Deadlines are ambiguous and do not have the intended meaning. It is not shown what happens to the interaction if a deadline is not met. Deadlines given raw values become unattainable in cases of iteration.

Ambiguities with using one timeline

Terminal messages and points are not obvious when using a single timeline. Most of the FIPA IPs do not check the conditions for continuing an interaction that may have already terminated earlier. In addition all the messages along a timeline can be sent concurrently, in spite of a decision depending on previous messages.

5.11.1 Propositions for Extended AUML

EAUML (Extended AUML), [Koning and al. 2001], is based on AUML with simplifications and modifications for control threads. Connectors in EAUML allow sending messages *n* times (broadcast), synchronisation (waiting for several messages to arrive), message triggering and causality. In message triggering, internal events trigger the message sending. Message causality indicates a causal relationship between two messages. Different versions of UML for agent interactions have been proposed but all of them are essentially AUML-like. The problems mentioned in the above sections for AUML still remain to be resolved.

Similar to the corrections in Figure 5.6 for the AUML diagram of the request protocol, the AUML notation should increase its expressiveness by annotating its protocols with states from messages being exchanged. In this way, states can be referred to as conditions and would solve some of the problems in AUML. Instead of

conditions occurring at only decision points, each message can be accompanied with its own condition or state.

The AUML notation may be enhanced by inserting users in the process and binding roles and including time and history management capabilities. AUML protocols may be validated through model checking techniques after being translated into *promela*, [Holzmann 1997], or Petri Nets. However at present there is no obvious and readable output from *Spin*, [Holzmann 1997], since it is not possible to translate the AUML protocol to an executable.

5.12 Petri Nets for Interaction Protocols

Petri nets [Petri 1966] is a graphical and mathematical modeling tool for describing and analysing information processing systems. Tokens are used to simulate and synchronise dynamic and concurrent activities. Algebraic equations can be derived from Petri nets. Petri nets are used in a variety of applications – e.g. communication networks, performance evaluation, manufacturing systems analysis and control, task planning, business process management and hardware design.

However a weakness of Petri nets is the complexity problem i.e. Petri-nets-based models tend to become too large for analysis even for a modest-size system [Murata 1989]. In applying Petri nets, it often necessary to add special modifications or restrictions suited to the particular application. Most Petri-net research groups have their own software packages and tools to assist the drawing, analysis and simulation of applications, [Feldbrugge and Jensen 1987].

5.12.1 Petri Nets Graphs

A Petri net is a directed graph with an initial state with two types of nodes: places (circles) and transitions (bars or boxes) with arcs (which may be weighed) between a place and a transition and vice versa. A marking assigns to each place a non-negative integer denoting the number of tokens. Places represent conditions, transitions represent events and tokens at a place represent the truth of the condition associated with the place. A transition has a number of *input* and *output* places. The dynamics

of a Petri net is a sequence of transition firing where tokens are taken away from input places to output places.

Synchronisation is needed when resources and information are shared among several processors and such synchronisation can be achieved through Petri nets such as mutual exclusion, readers-writers and producers-consumers problems. Synchronisation is needed when there is more than one input places. For example, in order for a transition to fire it has to wait for all the tokens to be present in the input places and then the transition is enabled. Transitions are active components that model activities which change the state of the system.

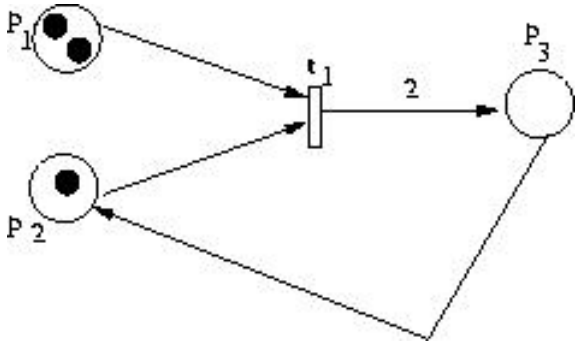


Figure 5.12 Example of a Petri net

For example in Figure 5.12, transition t_1 is fired when both places p_1 and p_2 are enabled. On firing, 1 token is removed from each p_1 and p_2 and 1 token added to p_3. Transition t_2 is fired when p_3 is enabled and transfers a token from place p_3 to place p_2.

Petri nets can be defined through set theory with tuples representing places, transitions and the relations between them (flow relations) (See [Murata 1989] and [Reisig 1985]).

5.12.2 Marked Petri Nets

Sometimes, k tokens indicate k data items or resources are available. A transition without input place is called a *source* transition and one without any output place is called a *sink* transition. For example, the chemical reaction $2H_2 + O_2 \rightarrow 2H_2O$ can be modeled as the following Petri net.

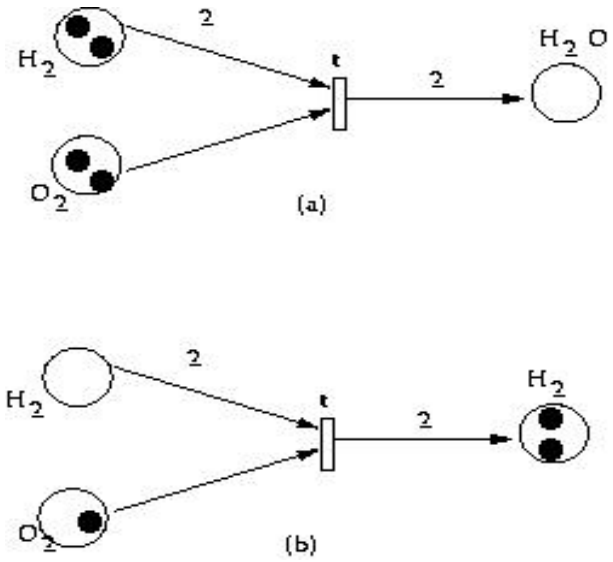


Figure 5.13 Example of a transition (firing) rule

Two tokens in each input place in Figure 5.13(a) show that two units of H_2 and O_2 are available, and the transition *t* is enabled. After firing *t*, the marking changes to the Petri net shown in Figure 5.13(b), where the transition *t* is no longer enabled. Usually there is a maximum to the number of tokens that each place can hold before/after a transition firing.

A state machine can be considered as an ordinary Petri net such that each transition has exactly one input and one output place. A marked graph has exactly one input transition and one out transition (thus no conflicts). In a free-choice net, every arc from a place is either a unique outgoing arc or a unique incoming arc to a transition (no confusion). There are extended free-choice net and asymmetric choice net. In contrast to a Petri net, a state machine does not have synchronisation.

5.12.3 Properties of Petri Nets

It is not obvious when conflicts will arise in Petri nets from just looking at a diagram and tools have been developed to detect such conflicts as deadlocks and other properties as liveness, performance evaluation and invariance satisfaction. Events that can occur concurrently may be mutually exclusive and are called *conflicting* events causing *confusion* situations. [Murata 1989] divides the properties of Petri Nets into two categories – those depending on the initial marking (marking-dependent or behavioural properties) and those that do not (structural properties). Example properties are:

- Reachability.
  A place p_2 is reachable from another place p_1 if there exists a sequence of firings leading from p_1 to p_2.
- Boundedness (or k-boundedness)
  The number of tokens in each place does not exceed a finite number k for any reachable marking.
- Safe
  A safe Petri net is said to be 1-bounded.
- Liveness
  Complete absence of deadlock. Although an important property, it may be costly to verify liveness in large systems. Thus liveness is relaxed to different levels of liveness.
- Reversibility
  The initial marking M_0 is reachable from any marking.
- Coverability, Persistence (for any two transitions, the firing of one transition will not disable the other) and fairness.

Methods for analysing Petri Nets may be classified into 1) the coverability (reachability) tree method (enumeration of all reachable markings – limited to simple nets due to the complexity of state-space explosion) 2) the matrix-equation approach and 3) reduction or decomposition techniques [Murata 1989]. The last two techniques are applicable to special subclasses of Petri nets in special situations.

A stochastic Petri net is a Petri net where each transition is associated with an exponentially distributed random variable that expresses the delay from the enabling to the firing of the transition. In case several transitions are simultaneously enabled, the transition that has the shortest delay will fire first. Such types of nets allow performance modeling.

5.12.4 Communication Protocols in Petri Nets

Petri nets can be used to represent and validate communication protocols [Billington and al. 1999]. The liveness and safeness properties of a Petri net are often used as correctness criteria in communication protocols.

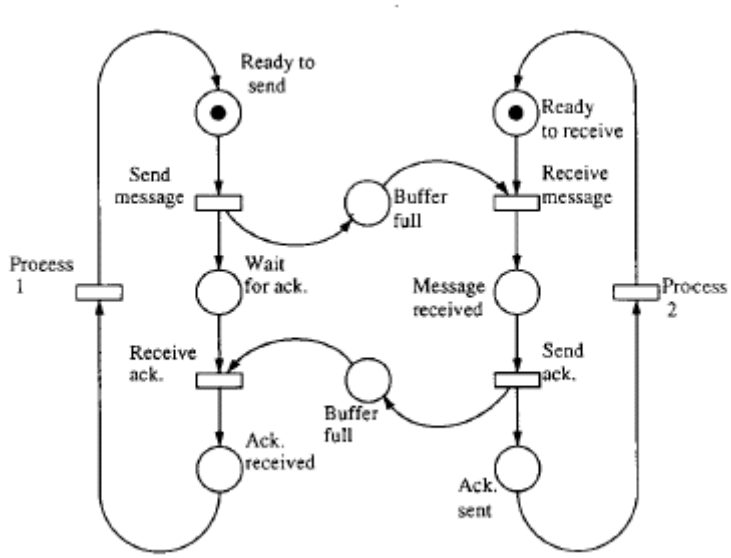


Figure 5.14 A simplified Petri net of a communication protocol [taken from Nowostawksi and al. 2001]

Figure 5.14 shows a simple communication protocol between two processes. Process 1 shows the sequence of actions of a message sender and receipt of an acknowledgment. Process 2 shows the sequence on the part of a message receiver and sending an acknowledgement. A buffer exists between processes 1 and 2. The Petri net is Figure 5.14 has been simplified but expressing richer or more detailed protocols in Petri nets would result in complicated diagrams.

5.13 Translating from Petri Nets to ANML

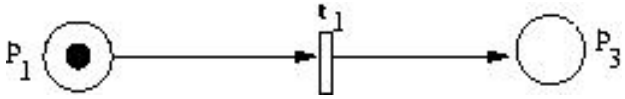


Figure 5.15 Simple Petri Net

Petri Net Entity	ANML Operator
Place	State
Transition	(Intermediate) state
Arc	Sub-process

The places and transitions in a Petri net can be regarded as states in ANML and the arcs between places and transitions are read as sub-processes. For example the Petri

net in Figure 5.15, p_1, t_1 and p_2 are states. Let the whole Petri net represent the process *browse*. The sub-processes from p_1 to t_1 is *browse-1* and the sub-process t_1 to p_3 is *browse-2*. Therefore the Petri net in Figure 5.15 is expressed in ANML as:

$p_1 \leftrightarrow [browse]\, p_3$

$p_1 \leftrightarrow [browse\text{-}1]\, t_1$

$t_1 \leftrightarrow [browse\text{-}2]\, p_3$

As can be seen, in Petri nets sub-processes are not explicit because the arrows (analogous to a sub-process) are not labelled by the name of a process. Rather in some Petri nets, the arrows are labelled with the number of tokens that fire a transition. It is possible to instead consider the transitions in a Petri net as processes e.g. in Figure 5.14, the transitions "send message", "receive ack.", "receive message" could be regarded as processes. However we can see that even places may be taken as processes i.e. places "ready to send" or "wait for ack". Moreover, processes can be synchronised at places e.g. in Figure 5.13 at p_1 and p_2 for transition t_1. In the Petri net in Figure 5.14, we do not find a clear distinction between the states and the processes apart from guessing from the names given to the places and transitions.

One feature of Petri nets lacking in statecharts is the synchronisation of processes through tokens before firing a transition. In ANML, this feature can be captured by extending ANML to deal with concurrency. In addition, synchronisation between different processes can be translated in ANML by different names assigned to different process instances. That is we can have different process instances in ANML through subscripts, for example $multi\text{-}lateral_{m1}$ and $multi\text{-}lateral_{m2}$ are two instances of a multi-lateral process – one with motion m_1 and the other with motion m_2). This is achieved by the ANML operator "::" i.e. $\alpha_1 :: \alpha_2$ which is read as process instance α_1 is constrained by process α_2. In this way, we associate sub-processes to their parent process. That is, a process may consist forking and merging of sub-processes where two or more processes are spawned by a parent process or two or more parent processes are joint to give a single process. It should be possible to retrace the roots of a child's process to its parent process(es). We capture the semantics of splitting and merging that occur in synchronised transitions firing through naming of sub-process threads. This may involve the creation of new processes (either through merging or splitting) which can be identified with their parent process through naming

mechanisms in ANML. Spurred by the remarks of the examiner, a more powerful ANML is discussed so as to embrace the synchronisation capabilities of Petri nets.

5.13.1 Merging of Processes

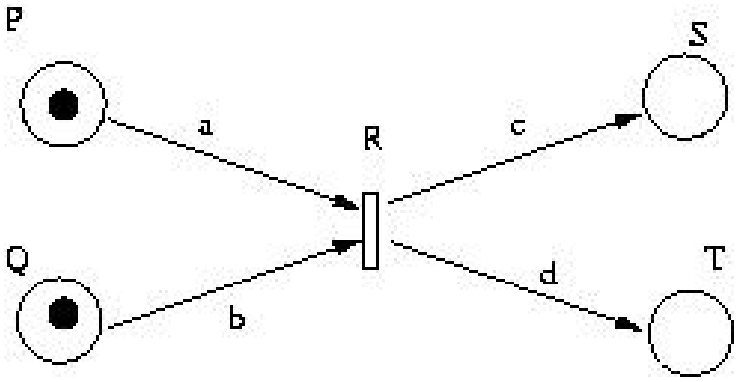


Figure 5.16 Merging of processes

Figure 5.16 includes the synchronisation and merging of two processes, from *P* and *Q* for transition *R* to fire. We can extend the ANML notation by dealing with synchronisation through an operator for concurrency borrowed from concurrent PDL [Goldblatt 1987]. Also, we use subscripts to name and associate the parent and children processes as they split or merge. More specifically, Figure 5.16 is expressed in ANML as:

$P_x \rightarrow [a_x]R_x$

$Q_x \rightarrow [b_x]R_x$

$(P_x \wedge Q_y) \leftrightarrow [a_x \cap b_y]R_{x+y}$

Merging of instances scripted with *x* and *y*

Concurrent and synchronised execution

The process $a_x \cap b_y$ denotes concurrent execution of processes a_x and b_y and the semantics of the $\cap$ operator are given in [Goldblatt 1987] for concurrent PDL. Moreover a_x and b_y are process instances of processes *a* and *b* and associated source states are P_x and Q_y. R_{x+y} denotes a named state of *R* and also denotes that state R_{x+y} emerges from source processes *a* and *b* (from the Petri net or ANML condition rule), more particularly the instances a_x and b_y (from $x+y$) and from the states P_x and Q_y.

5.13.2 Splitting of processes

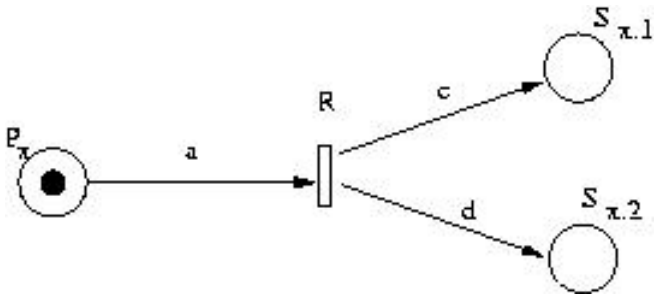


Figure 5.17 Splitting of Processes

After transition R is fired from P_x becoming enabled, two processes are created. These two process instances and the states they eventually lead to have to be related back to the parent process instance and source state P_z. For example in a multi-lateral negotiation, P_x may represent the state with motion x "to travel by train" and the process following this motion subsequently forks into two process instances c "walk to train station" and d "buy train ticket". The process instances c and d are related back to the motion x through subscripting them with x. In addition the states they lead to are explicitly named as having been triggered simultaneously from the same process instance a_x and state P_x. More specifically, Figure 5.17 shows that $S_{x\bullet 1}$ and $S_{x\bullet 2}$ are two branches emerging from the same parent process and are valid at the same time. Therefore the Petri net in Figure 5.17 is expressed in ANML as:

$$P_x \leftrightarrow [a_x]R_x$$
$$R_x \rightarrow [c_x]\, S_{x\bullet 1}$$
$$R_x \rightarrow [d_x]\, S_{x\bullet 2}$$

Forking into two processes and states

The state $S_{x\bullet 1}$ denotes that it is one branch from the source process a_x, child process c_x and state subscripted with x. The state $S_{x\bullet 2}$ denotes that it is another branch from the same source process (a_x) and state (R_x) as $S_{x\bullet 1}$, but with a different child process d_x. The merging and splitting Figure 5.16 can now be expressed as below in ANML:

$P_x \rightarrow [a_x]R_x$

$Q_x \rightarrow [b_x]R_x$

$(P_x \wedge Q_y) \leftrightarrow [a_x \cap b_y]R_{x+y}$

$R_{x+y} \rightarrow [c_{(x+y)}]\, S_{(x+y)} \bullet 1$

$R_{x+y} \rightarrow [d_{(x+y)}]\, T_{(x+y)} \bullet 2$

However Petri nets are not able to express alternative execution of processes as ANML does through the "∪" operator. That is, $R \leftrightarrow [c]S \cup [d]T$ cannot be shown in Petri nets notation. Similarly the above Petri nets do not show 1) the agents executing a process or 2) the roles of an agent, both which are needed for modeling agent interactions. Coloured Petri nets have been used in [Cost and al. 1999] for modeling agent conversations, as discussed in section 5.13.4.

5.13.3 Coloured Petri Nets

Coloured Petri nets is a class of Petri nets and allows synchronisation and resource sharing. It has been applied to model communication protocols. See [Jensen 1994, Jensen 1996] for an introduction to coloured Petri nets. As for the above (low-level) Petri nets, coloured Petri nets contain places, transitions, tokens and arcs describing the state changes. In coloured Petri nets, tokens carry data values which belong to a given type. Therefore tokens are distinguishable. Arbitrary complex data types can be also used as colour sets. Tokens that carry complex data values e.g. a list of many thousand records involving fields of different types. Colour set is treated as a type and token colour associated to token value. For a transition to fire, there must be sufficient tokens on its input places and these tokens must have token values that match the arc expressions.

Industrial applications of coloured Petri nets usually have 10-200 subnets with a total of 50-1,000 places and transitions and 10-200 types, [Jensen 1994]. Automatic verifications (such as model checking and invariants) and simulations (Design CPN [Jensen 1996}) of coloured Petri nets exist and can be used to investigate performance of a system.

5.13.4 Request Interaction Protocol in Petri Nets

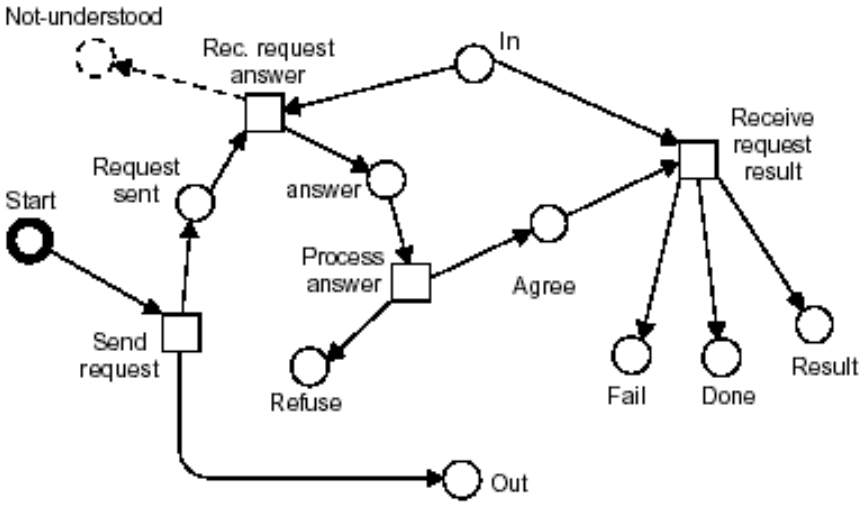


Figure 5.18 Petri net request conversation (Initiator Role), [taken from Purvis and al. 2002]

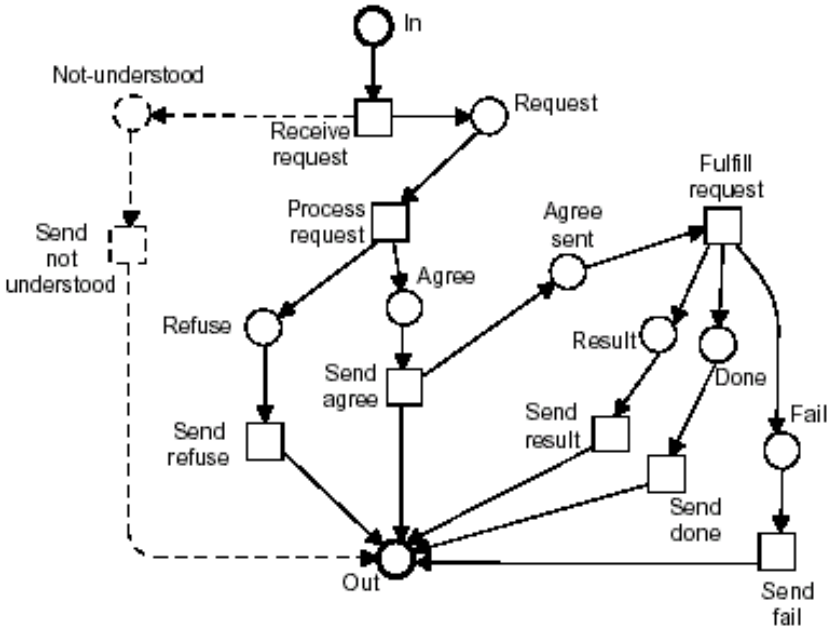


Figure 5.19 Petri net request conversation (Participant Role) [taken from Purvis and al. 2002]

Figure 5.19 and Figure 5.20 model the FIPA request IP [FIPA. 2001b], see section 5.5. There are two Petri nets – Figure 5.18 shows the initiator role and Figure 5.19 is used by a Participant. The collection of individual Petri nets associated with all the roles represents the entire interaction protocol [Purvis and al. 2002]. If there are more than 2 roles, then this can lead to issues about how the Petri nets are merged. In particular, it can be found that separating the Petri nets (Figure 5.18 and Figure 5.19)

for an initiator and participant yields two Petri nets that have similar (introducing redundancy) features that become implicit in the AUML or ANML joint protocol. Therefore translating from the AUML or ANML protocol requires added efforts for separating the nets and for replication of parts of a Petri net. There are a couple of notable issues in the above Petri nets for the Request protocol:

- As mentioned above, the separation of Petri nets for each role increases the complexity of a protocol which is supposed to be a *joint* series of actions. Compared to its AUML (Figure 5.5) or ANML (Theory 5.6) counterparts, Figure 5.18 and Figure 5.19 are more complex diagrams and harder to understand.

- The "In" place in the Initiator Petri net is unreachable in Figure 5.18 according to the Petri net notation. However the authors explain the "In" place to be a shared place common to two or more nets. This raises the issue of consistency of the "In" place, mutually exclusive access to it and whether it is necessary to separate the Petri nets. Still, a joint Petri could produce a complex and hard to read diagram as in Figure 5.20 for a Contract net protocol.

- In both Figure 5.18 and Figure 5.19, the Petri nets notation specifies that the places connected to a transition are all enabled after that transition fire. That is, there are only AND-communication messages. Hence in Figure 5.18, the "Agree" and "Refuse" places are both enabled implying that an agent both agrees and refuses after a "Process answer" transition. The same applies to an agent sending fail, done and request after a "Receive request result" transition.

- Similarly in Figure 5.19, the Petri net does not allow to express that after a "Fulfill request" transition, "Result" and "Done" places or processes are enabled/executed together or sequentially while "Fail" should only happen if "Result" and "Done" are not enabled.

- In Figure 5.19, after a "Send agree" transition, there should not be an arc leading to the "Out" place since this allows a premature end of the interaction.

5.13.5 Contract Net Interaction Protocol in Petri Nets

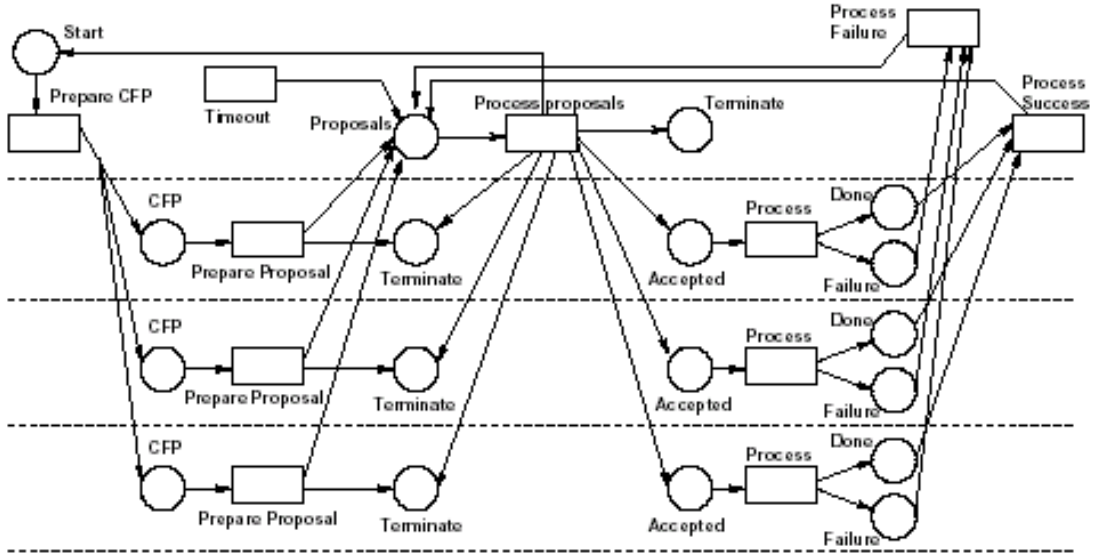


Figure 5.20 Contract Net conversation with 3 contractors [taken from Nowostawksi and al. 2001]

Figure 5.20 is a Petri net of a modified version of the FIPA Iterated contract net protocol in Figure 5.8. The Petri net does not include participants sending *not-understood* and *refuse-1* messages after receiving a *cfp* (a call for proposal). The Petri net has a *timeout* transition but this transition is not given any input and its output place is a proposal instead of an end place. The issues in Figure 5.20 are:

- The protocol ends in "Process Failure" and "Process Success" transitions rather than in terminal places.
- Showing 3 contractors in Figure 5.20 results in repeatedly expressing the same sequence of transitions and places three times for each contractor. This means redundancy in the Petri net and a significant increase in the degree of clutter to the Petri net compared to the AUML or ANML notation.
- The above point raises the issue that Petri nets are not scalable to richer protocols or for interaction between more agents. For example, adding the

understood and *refuse-1* messages of Figure 5.8 would render the Petri net illegible.

- The same point as in the Request Petri nets apply here in that all the places resulting a transition are enabled and the notation cannot show exclusive-or or alternative actions e.g. an *accepted*, *terminate* and new call for proposals all at the same time after a "Process proposals" transition.

5.13.6 A Pair-wise Negotiation Protocol in Coloured Petri Net

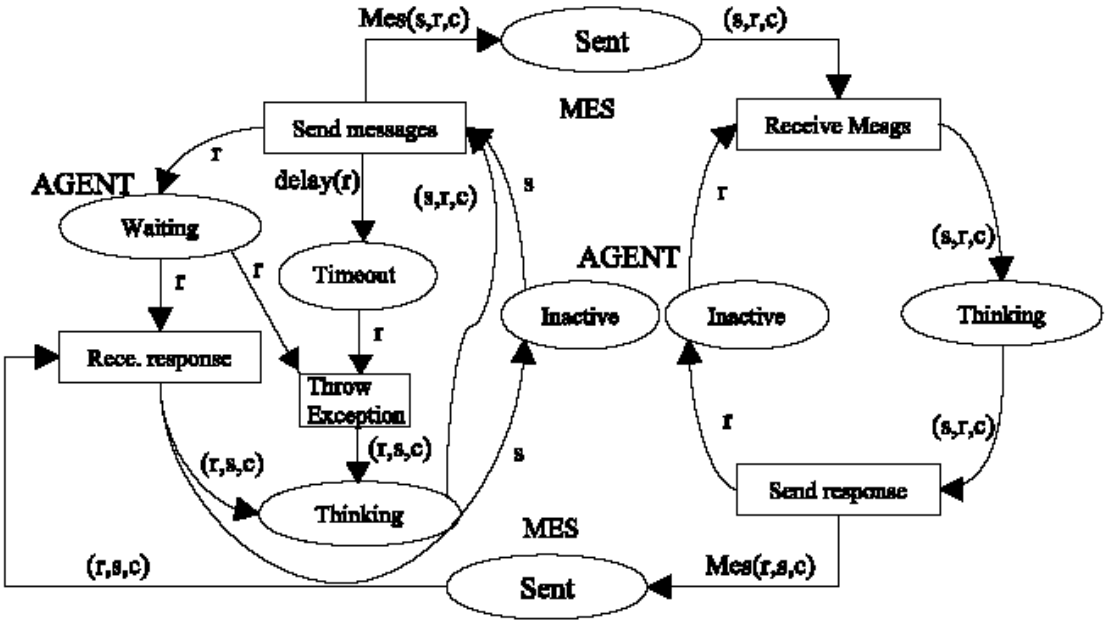


Figure 5.21 Pair-wise negotiation process for a MAS [Cost and al. 1999]

The coloured Petri net in Figure 5.21 presents a simple pair-wise negotiation protocol for two functional agents bargaining for goods, based on the FIPA ACL performatives set. There are 3 places: *Inactive*, *Waiting* and *Thinking*, which reflect the states of the agents. The buyer agent begins the negotiation (left hand side of Figure 5.21) and has the responsibility of handling message failures. Both agents are initially waiting in the *Inactive* places. The buyer initiates the negotiation process by sending a call for proposals (*CFP*) to some seller, and its state changes from *Inactive* to *Waiting*. The buyer is waiting for a response (*proposal*, *accept-proposal*, *reject-proposal* or *terminate*). On receipt, its state changes from *Inactive* to *Thinking*, at which point it must determine how it should reply. Once it replies, completing the cycle, it returns to the *Inactive* state.

- First there is no notion of a buyer or a seller executing an action or a process. Although Figure 5.21 is supposed to show the buyer and seller roles, it does not explicitly express which transitions and places belong to which roles.
- The labels AGENT and MES are given to some places but with no semantics of how these are related.
- It is not obvious where the protocol starts and ends.
- The protocol shows how an OS type messages are sent and received but it does not show the ACL performatives being sent and therefore does not represent an agent interaction protocol. The latter protocol is given in the English description but not in the Petri net in Figure 5.21.
- Figure 5.21 is essentially a flowchart with synchronisation between the transitions firing.
- The Petri net does not show that a buyer returns to an *Inactive* state once it replies as said in the informal specification. Moreover, it is not obvious where the cycle starts for (as claimed) the cycle to be completed.

As such, despite the above issues and no definition of the location of the token, we can express Figure 5.21 in ANML.

$negotiating \leftrightarrow$ one-of($\{open,\ timedout\}$)
$open \leftrightarrow sent \vee waiting \vee inactive \vee thinking$
$\neg negotiating \leftrightarrow$ none-of($\{open,\ timedout\}$)

$\neg negotiating \leftrightarrow [B{:}buyer.send\text{-}message]\ (waiting(B) \wedge sent(B))$
$sent(X) \leftrightarrow [timeout]timedout \vee [S{:}seller.receive]thinking(S) \vee$
$thinking(S) \leftrightarrow [S{:}Seller.send\text{-}response]\ (sent(S) \wedge inactive(S))$
$thinking(B) \wedge inactive(B) \leftrightarrow [B{:}Buyer.send\text{-}message_m \cap B{:}Buyer.delay_d]$
$(sent_m(B) \wedge waiting_m(B) \wedge timedout_d)$
$inactive(S) \wedge sent(B) \leftrightarrow [S{:}seller.receive]thinking(S)$
$waiting(B) \wedge sent(B) \leftrightarrow [B{:}buyer.receive\text{-}response](thinking(B) \wedge inactive(B)) \vee$
$timedout \wedge waiting(B) \leftrightarrow [B.throw\text{-}exception]\ thinking(B)$

Theory 5.12ANML Theory for Pair-wise Negotiation

5.13.7 KQML Register in Petri Net

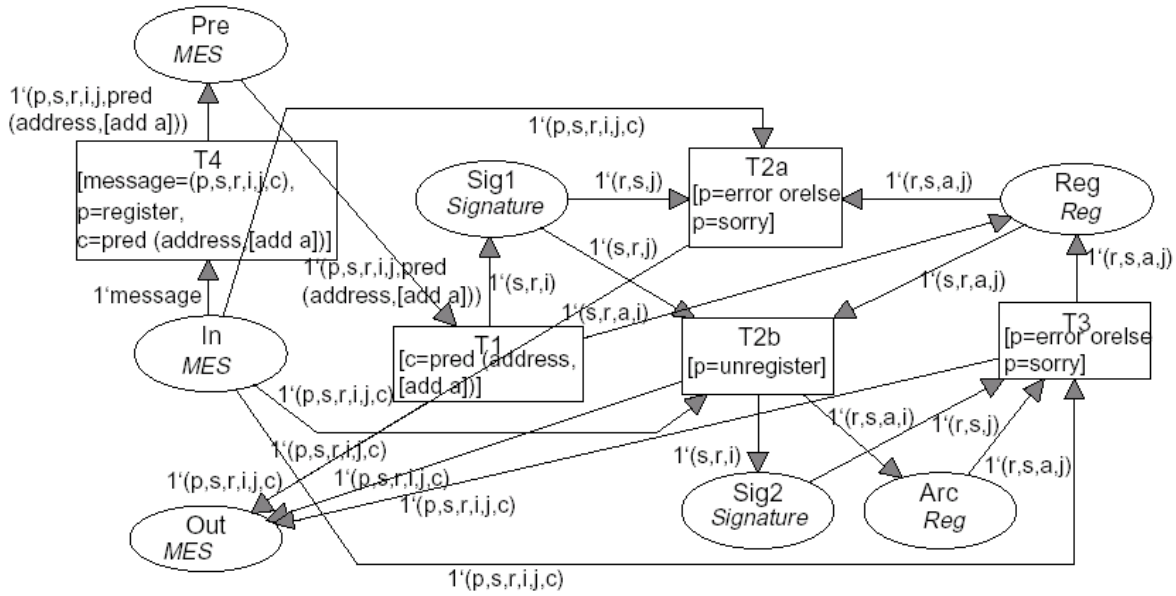


Figure 5.22 KQML Register

Figure 5.22 illustrates how the complexity of a Petri net can escalate from increasing the richness of a protocol. The figure shows the Petri net for a conversation for the KQML register performative. The places “Register” and “Done” serve as receipt locations for messages, after processing by the transitions *T1* and *T2*. A sender agent sends a message to a receiver with intermediate places to assure that the agents and ID fields in the response are in the correct correspondence to the initial messages. Figure 5.22 shows a Register conversation in coloured Petri net notation. A register consists of an initial “register” with no positive acknowledgement but a possible “error” or “sorry” reply. This registration may be followed by an unacknowledged “unregister”. See [Cost and al. 1999] for discussion about the coloured Petri net protocol in Figure 5.22. Here we introduce this diagram to show the complexity of a Petri net conversation centred around *one* KQML performative.

5.14 Issues about Petri Nets

5.14.1 Reachability Problem in Petri Nets

The reachability problem is one of the most important problems for Petri net analysis. It is also open to a large amount of variation in definition. Basically four related

reachability problems have been posed: 1) The reachability Problem 2) The submarking reachability problem [Wanatabe and al. 1989]. 3) The Zero-Reachability Problem (if the specific marking with zero tokens in all places is reachable). (4) The Single-Place Zero-Reachability Problem (if it is possible to empty all the tokens out of a particular place).

It has been shown that the reachability problem is decidable although it takes at least exponential space and time to verify in the general case. [Kosaraju 1982] and [Mayr 1984]. Reachability is an important problem, but not the only remaining problem for Petri nets. Liveness (related to deadlock) has received much attention in the Petri net literature.

5.14.2 Sub-Protocols

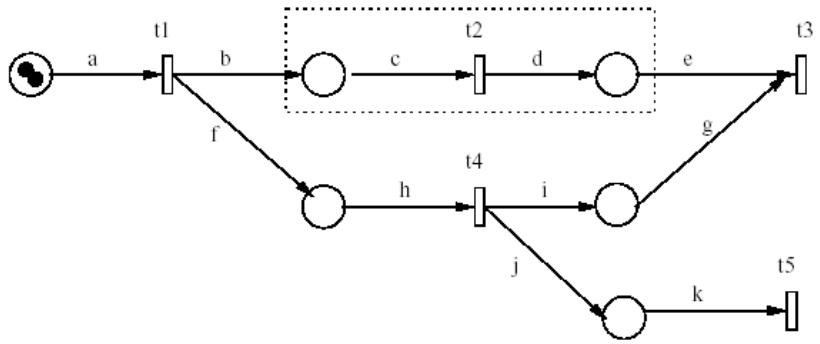


Figure 5.23 Example of an initial Petri net [Koning and Huget 2001]

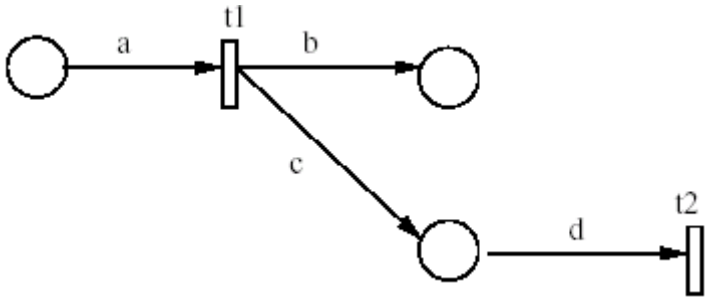


Figure 5.24 Piece of Petri net to add [Koning and Huget 2001]

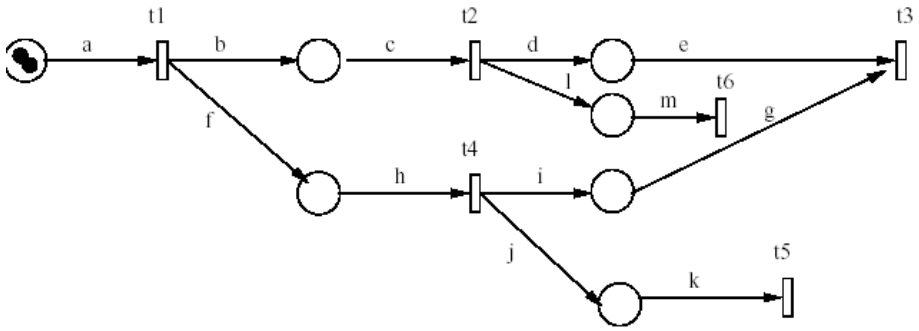


Figure 5.25 Resulting Petri net [Koning and Huget 2001]

Koning and Huget [2001] argue that Petri nets do not allow replacing a piece of protocol by another and thus are not suitable for reusability of protocols. For replacing of sub-protocols, Petri nets requires searching for the beginning and end of the parts to be replaced. They also claim that it is difficult to identify the exact semantics of these protocol parts. "For instance, in a high-level Petri net it is necessary to search for the place to be modified and to take the token into account since one cannot readily replace a portion of it by another. This poses some problems for firing transitions." [Koning and Huget 2001].

Combining protocols thus amounts to linking the output of one component to the input of the next one. Concatenating a high-level Petri net to another one imposes to take into account and probably modify the marking (the way tokens are placed) in order to prevent some transitions from becoming not fireable. For example, if an initial Petri net only bears m tokens whereas n are awaited ($n>m$), it becomes impossible to cross such transition. The problem is the same with colored Petri nets where variable names must be altered. [Koning and Huget 2001].

An example is given in Figure 5.23, Figure 5.24 and Figure 5.25. For the Petri net representation of a protocol given in Figure 5.23, replacing the doted area by the portion given on Figure 5.24 requires to modify the variables' name and change the tokens in order to produce the protocol shown in Figure 5.25. On the other hand, for a same interaction protocol defined in ANML, the hierarchy between states can allow abstraction of the state and easy replacement of a state.

5.14.3 Conclusions about Petri Nets for Interaction Protocols

The above analysis shows that Petri nets (including high-level or coloured) are not ideal for representing agent interaction protocols. The notation does not express hierarchy of states and thus does not facilitate reuse, abstraction or modification of protocols and sub-protocols as discussed in [Koning and Huget 2001].

Petri nets are found not to be scalable as can be seen in Figure 5.21 and Figure 5.22. Compared to the AUML or ANML specification of the Request protocol, its Petri net specification is more complicated. There is an issue of whether to separate the Petri nets according to the roles in the interaction leading to repeated specification of similar parts of a protocol. The issue also arises about shared places.

Processes are not explicit in Petri nets, but rather have to be deduced and named from the arcs between places and transitions. States and processes are not given equal status in Petri net notation and we cannot reason about the state of a negotiation as what the agents believe to hold at an instance of a negotiation. It is easier to understand states and processes than places that represent both.

As such the Petri nets have to be extended to the agents domain to show them performing processes, firing a transition or enabling a place. In [Cost and al. 1999], Figure 5.21, agents are represented vaguely as places, but without binding the agent to a particular instance or role.

Alternation or exclusion between executing processes are not captured with Petri nets notation. For example execution of either one of two or more transitions (process alternation) leading to the same state. In another case, Petri nets cannot show that an agent can only choose and execute one process among several possible ones.

Section 5.13 shows how synchronisation between processes as possible in Petri nets are captured in an extension of ANML. The processes that merge or fork are related to the parent processes.

A class of problems arises from optimisation considerations for Petri nets. If a Petri net exhibits a certain behaviour, as indicated by its set of transition firing sequences and its reachability set, can the Petri net be changed (optimised) without affecting its behaviour? This may involve deleting dead transitions (which can never be fired) and dead places (which can never be marked) or perhaps the redefinition of some transitions and has led to a number of related work.

5.15 Summary

In this chapter, existing protocols are verified and their corrected versions represented in ANML. It is found that the AUML notation is an error-prone, informal and ambiguous methodology for representing protocols. The complexity of AUML IPs increases drastically for even simple protocols like the bilateral protocol. AUML is not suitable for representing *n-m* interactions with time and history management in a multi-agent system. The corresponding protocols given in the ANML meta-language has been subject to a number of verification cycles for correctness. Then protocols in Petri nets notation are analysed and compared with ANML protocols. There is no doubt that there are other effective notations like Petri nets, but we do not follow them because they are far from the logical work of interest in this thesis.

In addition to functional correctness, a logical theory for a protocol enables us to address properties such as termination, fairness, safety and liveness. The next chapter analyses protocols for these latter properties.

6 The Properties of a Protocol

6.1 Introduction

As shown in the previous chapter protocols must be verified for correctness. In addition, interaction protocols can be designed and analysed for other desirable properties such as safety and liveness. A safe and sound interaction allows no unpredictable states and transitions, and negotiation states that become valid are those that are defined by the protocol and none other. A liveness property asserts that a process eventually enters a desirable state and that there are no deadlocks. There is a relationship between liveness, termination, soundness and safety properties. A safety property asserts that nothing bad happens and termination asserts that something happens so both can be used to partly derive liveness.

Expressing a protocol as a logical theory enables the proof of its correctness and its properties. This chapter focuses on specifying and proving safety and liveness properties of negotiation protocols expressed in ANML, [Paurobally and Cunningham 2002b]. The properties of a protocol are expressed as axioms in ANML, enabling to check whether a property holds in the logical theory of a protocol. Axioms are defined for *termination*, *soundness*, *liveness*, *serialisability*, *completeness*, *security*, *reliability*, *fairness*, *decidability* and *stability* properties. These properties can be used to choose between protocols or a protocol amended to exhibit given properties. Termination and soundness are sub-features of the safety property. Possible sequences of actions are analysed in order to prove liveness. A negotiation terminates if it can be proved that all paths in a protocol lead to a terminal state without getting stuck in an infinite loop. A safe and sound negotiation contains no unpredictable states and actions and states that become valid are those that are defined by the

protocol and none other. Liveness and termination properties cover the absence of deadlock and livelock. As an example, the properties of the bilateral protocol, specified in previous chapters, are proved through mostly structural induction, though other methods such as invariants and proof lattices may also be used, [Keller, 1876; Lamport, 1980; Sistla 1994].

6.2 Safety and Liveness properties – A survey

Safety and liveness properties, [Lamport 1977], have been extensively studied in the verification of concurrent programs. A safety property asserts that something bad never happens, i.e. the program never enters an unacceptable state. A liveness property expresses that something will eventually happen i.e. the program eventually enters a desirable state. Other properties may be classified into safety and liveness properties, which allows for choosing a suitable proof method to show that a program exhibits a particular property. Alpern and Schneider, [Alpern and Schneider 1985], give a formal definition of both safety and liveness properties, where a decomposition theorem proves that every property can be represented as the conjunction of a safety and a liveness property. Safety properties include partial correctness, mutual exclusion and absence of deadlock. Partial correctness states that from a true precondition a process never terminates with a false post-condition, [Owicki and Lamport, 1982]. Partial correctness may be the only safety property that is required in sequential programs. Mutual exclusion means that two different processes are never both in their critical section where only one process can be. A liveness property includes program termination which is usually proved through induction. Liveness may also mean absence of livelock or the existence of cyclically recurring states. Fairness is an important aspect of liveness in concurrent processes. It implies that every process gets a chance to make progress. Sistla, [Sistla 1994], introduces the notions of strong safety and absolute liveness. Strong safety indicates a safety property which is closed under stuttering i.e. the property is insensitive to successive repetition of any state of a sequence. [Chang and al. 1992], propose a classification as a hierarchy of properties where the higher levels provide the finer distinction of different liveness properties. Their approach is motivated by the different techniques for proving the properties at the different levels, [Kindler 1994].

Different techniques are used to prove safety and liveness. Owicki and Lamport, [Owicki and Lamport, 1982], use proof lattices or well-founded induction to prove liveness properties. Methods based on global invariants have been used to prove safety properties, [Manna and Pnueli, 1995]. Temporal logics has been proposed by Pnueli, [Manna and Pnueli, 1995; Pnueli, 1977], for the specification and verification of concurrent programs since it provides a convenient language for expressing properties of reactive systems. A safety property may be expressed by a formula such as $\Box p$, [Manna and Pnueli 1995], i.e. p is true at every accessible state of the system. This has been followed by several versions of temporal logics in the field, [Burstal, 1974; Lichtenstein and al. 1985; Manna and Sipma, 1998; Sistla and Clarke, 1985]. Other verification techniques for proving properties include deductive verification, [Gordon and Melham, 1993], verification diagrams, [Bjorner and al. 1999], model checking and abstraction, [Bjorner and al. 1999; Shostak, 1984]. Model checking tools can verify properties of finite-state systems while deductive tools verify infinite state systems. Verification rules or model checking may use theorem-proving and invariants to prove temporal properties. Abstraction allows proving properties over an abstract system and then inferring the validity of a related property over a larger or more complex concrete system, [Bjorner and al 1999].

6.3 Some General Properties of Protocols

In transition systems, a state is an interpretation of system variables and a transition is a function mapping a state into a set of successor states. A computation is an infinite sequence of states. Computational methods are used for proving a protocol's properties.

6.3.1 Safety

One or more safety properties must be proved in order to prove a liveness property. Proofs by induction or invariants are mostly used, though other methods may be used, [Lamport 1980a, Keller 1976, Sistla 1994]. Any assertion that describes a superset of the reachable states of a system may be an invariant. To prove an invariant, which is a condition or state of affairs, it can be shown that at state s_1, the invariant is true and execution of an atomic action from s_1 leaves the invariant true. This chapter adopts such an approach for proving some of the properties of a protocol. Work on

generating invariants can be found in [German and Wegbreit 1975]. A property expressed as the formula □*p* is an invariant if *p* is an assertion.

In ANML, a safety property is defined as $P \rightarrow [\alpha]Q$, where *P* and *Q* are assertions about the state of the system or negotiation and *Q* is the invariant. If the negotiation starts with *P* being true, then *Q* is always true after any action. Proving that *Q* is an invariant requires reasoning about programs. This can be proved inductively by proving that the invariant *Q* holds initially and is preserved by every atomic action and hence transition in the ANML protocol. Executing a single atomic action yields a new state in which either the invariant is true or the safety property does not hold. If *N* is the entire negotiation, then $I \rightarrow [N]\ I$ means that *I* is an invariant that is true before and after the negotiation.

6.3.2 Partial Correctness

Partial correctness means that if an interaction starts with a precondition being true, it can never terminate with a false postcondition. In an ANML protocol, the initial state before starting a negotiation is ¬(*outermost parent state of the negotiation*). e.g. The overall parent state a bilateral negotiation is *negotiating* which contains *open* and *closed* as sub-states. The parent state of all states in a multilateral negotiation is *multi-lateral*. The postcondition for a negotiation usually consists (not entirely) of ending in a terminal state e.g. *closed*.

In ANML, partial correctness then allows statement of: ¬(*parent state*) ↔ [*Agent-group*. *negotiation-process*] *terminal state*. For example, in a promissory negotiation *p* between agents *X* and *Y*, partial correctness would be ¬*promised* ↔ [{*X*,*Y*}.(*p*: *promissory-negotiation*(¬*promised*)] *p*>>*closed*.

6.3.3 Liveness

In order to exhibit liveness, a negotiation must either eventually terminate after a process or the entire negotiation halts in a state in which execution of a finite number of actions complete the negotiation. Liveness can be inductively proved by assuming that atomic actions always terminate and that after executing a process, it is always possible to complete the execution of a negotiation. Liveness entails that if a

negotiation reaches a certain state, then any atomic action executable from that state can eventually be performed and terminated. The states and actions are defined in the protocol.

6.4 Definition of Properties

This section defines properties that may be useful in choosing between protocols by analysing how transitions between states satisfy or violate such properties. Ten possible properties of a protocol are presented: *termination*, *safety* and *soundness*, *liveness*, *serialisability*, *complexity*, *completeness*, *security*, *reliability*, *consistency* and *decidability*. The list is by no means complete and protocol designers may specify and prove other properties for their protocols.

An ANML protocol may be represented as a graph with states as nodes, atomic processes as edges and a compositional process as a path. An ANML protocol is analogous to a directed graph of possible states and transitions towards a terminal state. An execution of a negotiation process may follow a subset of those paths given by its protocol. Given the theory of a protocol, the possible states and paths can be logically derived.

6.4.4 Preliminary Definitions

This section provides preliminary definitions of sets of states and transitions that are used for specifying and proving properties of protocols. From the syntax of ANML in chapter 3, let the following notations hold:
The set ϕ denotes the set of all propositions, ω denotes the set of all atomic processes and *Agents* denote the set of all agents. (See chapter 3 for the syntactic definitions of these sets.) $A \in Fma(\phi, \omega, Agents)$, $\alpha \in Proc(\phi, \omega, Agents)$ where $Fma(\phi, \omega, Agents)$ is the set of formulae and $Proc(\phi, \omega, Agents)$ is the set of processes. Agents denoted by X, Y, X_i, Y_i are of type *Agent_group*.

Let ε denote the set of all negotiation states A given by the function $Fma(\phi, \omega, Agents)$ and let ψ represent the set of all processes α given by $Proc(\phi, \omega, Agents)$. That is let $\varepsilon = Fma(\phi, \omega, Agents)$ and $\psi = Proc(\phi, \omega, Agents)$. Let μ represent the theory of a protocol in ANML.

$state\text{-}set = \{s,s_1: \varepsilon \mid \mu \vdash ((s \vee (s_1 \rightarrow s) \vee (s \rightarrow s_1)) \bullet s \}$

$action\text{-}set = \{s, s_1: state\text{-}set, t: \omega \mid \mu \vdash (s \rightarrow [t]\, s_1) \bullet t\}$

$path\text{-}set = \{s, s_1: state\text{-}set,\ p: \psi \mid \mu \vdash (s \rightarrow [p]\, s_1) \bullet p\}$

$sub\text{-}path(p_1, p_2) \leftrightarrow (\exists\, p_3\,,\ p_4 \in \text{path-set}\ (p_4; p_1; p_3 = p_2))$

The notation $ASet = \{x, x_1, \ldots: Type \mid cond \bullet x \}$ is read as *ASet* is a set where if $x, x_1, \ldots$ are of type *Type* (e.g. real numbers) and the condition *cond* is true, then the elements of *ASet* are x.

The set *state-set* denotes the set of negotiation states possible in protocol μ, containing parent or sub-states. The set *action-set* denotes the set of atomic transitions defined in the protocol μ. A process is a path containing allowed transitions. The set *path-set* denotes the set of possible paths in protocol μ. The set of paths, *path-set*, consists of the processes in *action-set* related with ANML connectors through the function *Proc*(*state-set*, *action-set*, *Agents*). It is useful to define the function *sub-path* which returns *true* if p_1 is a sub-path of p_2. The above definitions are used in the rest of this chapter for defining properties that can be exhibited by a protocol.

The following predicates are defined since they also are useful for specifying and proving the properties of a protocol. The predicate *terminating*(s_t) returns *true* if s_t is a terminal state, i.e. no actions are possible from state s_t.

$terminating(s_t) \leftrightarrow (\neg\exists\, p: path\text{-}set,\ s_i: state\text{-}set\ ((s_t \rightarrow [p]s_i) \wedge \neg(s_t \leftrightarrow s_i))$

where $s_i, s_t \in state\text{-}set$

The formula $(\neg\exists\ p: path\text{-}set,\ s_i: state\text{-}set\ ((s_t \rightarrow [p]s_i) \wedge \neg(s_t \leftrightarrow s_i))$ states that no actions are possible from the terminal state s_t. There does not exist a path p that leads from state s_t to s_i.

The predicate *outermost* (s_0) returns *true* if the state s_0 is the overall parent state.

$(outermost(s_0)$ iff $\forall s \in state\text{-}set\ \mu \vdash (s \rightarrow s_0))$

An entry process, p_e, in a protocol μ can only be performed when the outermost state is *false*. The predicate *entry*(p_e) returns *true* if p_e is an entry process.

$$\forall s_0, \exists s_1 \in \textit{state-set}, p_e \in \textit{path-set}\ (\mu \vdash (\textit{outermost}(s_0) \wedge \neg s_0 \rightarrow [p_e]s_1) \rightarrow \textit{entry}(p_e))$$

6.4.5 Termination Property

Termination analysis, [Codish and Taboc, 1999], attempts to determine whether evaluation of a given expression will definitely terminate. The evaluations of a constant or of a non-recursive function terminate, in the case where their sub-procedures terminate. A recursive function can be shown to terminate if it can be proved that there is a base case that will inevitably be executed. Termination analysis gives either a program "definitely terminates" or "don't know". There has been increasing work in the automatic termination analysis of logic programs, such as transforming a consistent logic program into a term rewriting system and proving termination for the latter, [Ganzinger and Waldmann, 1993; Ohlebusch 1999]. Herein, a protocol terminates if it can be proved that a negotiation process complying with that protocol can eventually reach a terminal state. To do so, it is proved that not only there is a path leading to a terminal state from all states in a protocol, but also the stronger requirement that all possible paths lead to a terminal state. A terminal state may be the sub-state of a parent terminal state e.g. the state *rejected* is a sub-state of *closed*. Therefore all paths must lead to the parent terminal state in a terminating protocol. The predicate *terminal-parent*(s_{pt}) returns *true* if s_{pt} is the overall parent terminal state i.e. *closed* in some protocols.

$$\textit{terminal-parent}(s_{pt}) \leftrightarrow (\textit{terminating}(s_{pt}) \wedge \forall s\colon \textit{state-set}\ (\textit{terminating}(s) \rightarrow (s \rightarrow s_{pt}))$$

The predicate *terminal-parent*(s_{pt}) returns *true* when the state s_{pt} is a terminal state and all other terminal states are sub-states of s_{pt}.

Termination Property for protocol μ:

$$\mu \vdash \forall s, s_{pt}\colon \textit{state-set}\ \exists\ p_t : \textit{path-set}\ (\ (\ \textit{terminal-parent}(s_{pt})) \rightarrow (s \rightarrow [p_t]\ \ s_{pt}\ )\ )$$

The termination property states that from all states s in a protocol μ, all possible paths p_s lead to a parent terminal state s_{pt}, by following p_s with p_t. It is assumed that an atomic transition in an action-condition rule terminates by nature of being atomic.

A stronger termination property specifies that all executions reach a parent terminal state, without getting stuck in an infinite communication loop. The strong termination property holds if the above *termination property* holds and no states is revisited infinitely.

Strong Termination Property for protocol μ:

$$\mu \vdash \forall s, s_1 : \textit{state-set}\ \neg\exists\ p : \textit{path-set}\ (s \rightarrow [p^{>1}]\ s_1)$$

where $s_i, s_t \in \textit{state-set}$

6.4.6 Soundness

A safety property asserts that a program never enters an unacceptable state. Soundness can be considered as a sub-property of safety and implies that a process terminates properly and there are no references to undefined states. A safe and sound negotiation is defined as containing no unpredictable states or actions and the negotiation states that become valid are those that are defined by the protocol and none other. Valid negotiation states mean the current states of negotiation which are *true* at that point and are known by all participants to be *true*. All states other than the current state and its parent states are *false*. All states and allowed actions are well-defined and as expected at all instances. For example, if there is a model that gives blue and red pens, then the soundness property states that only red and blue pens are obtained and no others.

Soundness property for protocol μ: $\forall\ A : \varepsilon\ \ ((\mu \vdash A) \rightarrow (\mu \vDash A))$

If the protocol μ is sound, then a negotiation instance δ does not yield any states, processes or transitions other than those defined by μ.

The proof sequence ($\mu \vdash A$) defines a possible state or state transition inferred in a negotiation instance and this is valid only if it is entailed by the model of the protocol μ. A process p in a state transition A could be a *null* process in which case the formula A defines a relation between states and sub-states.

The soundness property between a protocol, μ, and a process, δ, which is an instance of μ, can be clarified as below:

$$\mu \vdash (s_0 \leftrightarrow ([a_1]s_1 \vee \ldots \vee [a_n]\, s_n)) \rightarrow \neg(\delta \vdash (s_0 \rightarrow([a_z]\, s_z))$$

where $z > n$)

for all states s_0 to $s_n \in$ *state-set*, a_1 to $a_n \in$ *path-set*, s_z: ε and a_z: Ψ

Soundness ensures that only the actions defined by a protocol may be executed and none other. The rule $s_0 \leftrightarrow ([a_1]s_1 \vee \ldots \vee [a_n]s_n$ gives the possible state transitions from state s_0. The implication $\neg(\delta \vdash (s_0 \rightarrow ([a_z]s_z))$ where ($z > n$) ensures that no transitions other than those defined by the protocol μ are possible in process δ. An example of a protocol not being sound is if it asserts that *rejected* and *agreed* are mutually exclusive states and yet a negotiation instance has both *agreed* and *rejected* as *true*. A negotiation which is in the state *xanadu* where *xanadu* is undefined by its logical theory may be due to an unsound protocol.

6.4.7 Liveness

A liveness property asserts that something eventually happens and that is there is no deadlock. Safety properties describe allowed behaviour while liveness properties describe required behaviour. A liveness property and termination ensure the absence of deadlock and livelock. In a non-deadlocked process, after an action *X*, another action *Y* can occur to change a non-terminal state. To prove liveness and the absence of deadlock, the behaviour of an execution is defined and the possible sequences of actions are analysed. In a protocol, an agent either starts the negotiation process or responds to a previous action, both of which may possibly change the current state and produce a deadlock. In case of deadlocks, ANML easily allows addition of the occurrence of internal events such as *timeout* or *overdue*.

Liveness property for protocol μ: $\mu \models \forall\, s_i$:*state-set*, a_i :*path-set* $(s_0 \leftrightarrow [X_1.a_1]s_1 \vee \ldots. \vee [X_n.a_n]s_n) \rightarrow \neg\exists\, b$: *path-set* (($(Y.b\,?; X_i.a_i) \wedge (X_i.a_i\,?; Y.b)$) $\vee$ [*timeout*]Z)

where $0 \leq i \leq n$, Y and $X_i \in A\_group$

A liveness property asserts that it is not the case that an agent (or a group of agents) X waits for another agent (or group of agents) Y to proceed while Y is waiting for X. Otherwise there is deadlock and none of the agents are able to proceed in their execution since they are waiting for each other. In the formula (($Y.b$?; X_i .a_i) $\wedge$ ($X_i.a_i$? ; $Y.b$)), to do process a_i, agent X_i must wait for agent Y to perform process b. At the same time, for agent Y to perform process b, it must in turn wait for agent X_i to perform process a_i. The only option would then be termination of the process through a *timeout* event. A protocol μ satisfies the liveness property if the *liveness* property applies for all possible states, *St.*, and paths *Pt.* in that protocol where *St.* $\in$ *state-set*, *Pt.*$\in$ *path-set*. There can be additional constraints to deadlock avoidance when defining the liveness property.

6.4.8 Serial, Ordered and Serialisable

The serialisability property originates from multi-user database transactions where entire transactions are treated as single atomic units with respect to each other. The atomicity of transactions is enforced by ensuring that interleaved execution of concurrent transactions remains serialisable, [Eswaran and al. 1976]. A schedule S is *serial* if, for every transaction T participating in the schedule, all of T's operations are executed consecutively in the schedule; otherwise it is called *non-serial*. This is equivalent to a path consisting of a sequence of sub-processes. A schedule S of n transactions is serialisable if it is equivalent to some serial schedule of the same n transactions.

Serial property for protocol μ:

$\mu \vdash \forall\, s_i$:*state-set*, a_i :*path-set* $(s_0 \rightarrow [a_1;\ldots; a_n]s_n \wedge \neg sub\_path(p_i\,, p_k))$

$\rightarrow\ \neg(\, s_i \rightarrow [p_i]\, s_j \wedge\ [p_k]s_k\,)$

where $(0 \leq i, j, k \leq n)$

for all states s_0 ,s_1, s_p , s_q $\in$ *state-set*, and a_1 to a_n , p_1 to $p_n \in$ *path-set*.

A protocol is serial if all its derivable paths are executed sequentially and there are no parallel executions.

A protocol is *ordered* if the sequential processes in a path cannot be out of order or in parallel. If from state *s*, agent *X* executes a process followed by agent *Y*, then the protocol is ordered if it is not permitted for *Y* to execute its process before or in parallel with *X*.

There is a similarity between non-serialisable database transactions and ordered paths in a negotiation. In both cases, there is a sequence of execution that needs to be respected so as to safeguard the integrity of the overall process. The ordered property is respected if none of the sequential processes can occur in parallel.

Ordered property for protocol μ:

$\mu \vdash \forall\ s_i$:*state-set*, a_i :*path-set* $((s_0 \rightarrow [a_1;\ldots; a_n]s_1) \rightarrow \neg\ (s_0 \rightarrow [a_i ; \ldots.; a_j]\ s_1)$

where $(0 \leq i,j \leq n)$ and $i\ ..j$ is not equivalent to $1..n$

for all states s_0 to $s_1 \in$ *state-set*, and a_1 to $a_n \in$ *path-set*.

The formula $(s_0 \rightarrow [a_1;\ldots; a_n]s_1$) entails that the execution of the path $p = (a_1;\ldots; a_n)$ leads from state s_0 to s_1. The formula $\neg(s_0 \rightarrow [a_i ; \ldots.; a_j]\ s_1)$ implies that the sub-processes of the path *p* cannot be executed in any other order.

Serialisability may be considered as contrary to the ordered property since a path $(a_1;\ldots; a_n)$ is serialisable iff there exists another path equivalent to it but with $(a_1;\ldots; a_n)$ in a different order.

Serialisability property for protocol μ:

$\mu \vdash \forall\ s_i$:*state-set*, a_i :*path-set* $((s_0 \rightarrow [a_1;\ldots; a_n]s_1) \leftrightarrow (s_0 \rightarrow [a_i ; \ldots.; a_j]\ s_1)$

where $(0 \leq i,j \leq n)$ and at least one of the numbers in $i..j$ is not equivalent to its corresponding number in $1..n$

for all states s_0 to $s_1 \in$ state-set, and a_1 to $a_n \in$ path-set.

6.4.9 Completeness

In software engineering, *completeness* defines the degree to which all the parts of a software system or component are present and each of its parts is fully specified and developed, [Boehm and al 1978]. In state automata, completeness depends on the extent of possible inputs for which the output is defined. In practice, completeness of a system is defined with respect to an application. An example is the case of a failure detector. The failure detector satisfies a completeness property if it detects whenever a process is faulty. There are cases of strong and weak completeness where strong completeness means that eventually every process that crashes is permanently suspected by all detectors. Weak completeness entails that eventually every process that crashes is permanently suspected by some detector, [Panconesi 1997]. Completeness of a system A with respect to model B occurs if A can reach every conclusion which is true in B. The dual to completeness is soundness. If a model yields red and blue pens, the completeness property ensures that the model will give both red and blue pens.

In an ANML theory of a protocol, the completeness property asserts that at any point the truth value of a state and state transitions are either true or false and thus defined at all times. If a protocol, μ, declares states or paths to be possible then in a negotiation instance, γ, the states and paths and their implications are possible.

Completeness property for protocol μ: $\forall A$: ε ($\mu \models A \rightarrow \mu \vdash A$)

Since it is not possible to have undefined states in a complete protocol, the formula (($s \vee \neg s$) $\wedge$ (($s \rightarrow [p]\ s_1$) $\vee \neg(s \rightarrow [p]\ s_1)$) would not hold, where s and s_1 are states and p is a process. If a protocol, μ, allows a formula to be valid, then it holds also in a process, γ, i.e. $(\mu \vdash A) \rightarrow (\gamma \vdash A)$.

6.4.10 Non-Arbitrary Restart

Non-arbitrary restart is a sub-feature of the safety property. This property asserts that a negotiation cannot be arbitrarily restarted. Therefore a protocol is secure if entry processes can only be performed before starting a negotiation. Some protocols allow

restarting from a specific state. The outermost state of a process is always *true* once the process has started and is *false* before starting a process. All other states in a protocol are sub-states of the outermost state. The state s_0 is the overall parent state if *outermost* (s_0). An entry process, p_e, in a protocol μ is possible when the outermost state is *false*. The predicate *entry*(p_e) returns *true* if p_e is an entry process.

Non-arbitrary restart Property for protocol μ:

$$\neg\exists s_i, s_j \in \textit{state-set}, \forall p_e \in \textit{path-set}(\mu \vdash (\textit{entry}(p_e) \wedge \neg \textit{outermost}(s_i) \wedge s_i \rightarrow [p_e]\, s_j))$$

The above property asserts that it is not possible to execute the entry process p_e from the state s_i, where s_i is not the outermost parent state. Parameterisation of states ensures that processes are executed by the required participant. For example, *X* cannot *agree* to its own *offer* in a bilateral negotiation and this is enforced by a protocol.

A protocol that never allows an agent to perform two consecutive actions without a transition from another agent could include the following rule:

Non-consecutive property for protocol μ:

$$\forall s, s_0, s_1 \in \textit{state-set}, p, p_0 \in \textit{path-set}\ (\mu \vdash (s_0 \rightarrow [Y.p_0]\, s\,) \wedge (s \rightarrow\ [X.p]s_1) \rightarrow \neg(X{=}\ Y))$$

6.4.11 Reliability

A system is reliable if it consistently produces the same results, preferably meeting or exceeding its specifications. There are several approaches to classifying the reliability properties of real-time systems e.g. ultra-reliable and fault-tolerant systems, hazard analysis and system safety or the distinction between security and safety, [Rushby 1994]. In protocol design, reliability can be defined as an extension of safety, security and liveness properties. A reliable protocol entails that a negotiation process γ will eventually achieve its goal. Therefore, in a reliable protocol, there is a possible path to a goal state *g*, yielding reachability of the goal state.

Reliability Property for protocol μ:

$$\forall s,g\text{: } state\text{-}set(\ \mu \vdash \exists p\text{:}path\text{-}set\ \ (goal(g) \rightarrow\ (s \rightarrow <p>g)))$$

The predicate *goal*(*g*) means that state *g* is a goal state. The formula ($s \rightarrow <p>g$) is read as in the negotiation process, there is a possible path *p* to the goal *g* from any state *s*. If $\neg s$ is the state of a negotiation at the start, then all possible paths are equal to or a subset of path *p*. If path *p* can lead to goal *g*, this implies that there is a possible path leading to goal *g*.

6.4.12 Consistency

Consistency means that formulae *A* and $\neg A$ cannot both be valid. A protocol is consistent when for all states *A,* either *A* or $\neg A$ is true and it is never the case that ($A \wedge \neg A$). A consistent protocol contains no contradiction or undefined states.

Consistency property for protocol μ: $\forall\ s \in state\text{-}set\ (\ \neg\ (\mu \vdash (s \wedge \neg s))$.

6.4.13 Non-Concurrent Processes

A process is concurrent when two states that do not have the same direct parent can be *true* simultaneously. A process is not concurrent if two states without the same direct parent cannot be *true* simultaneously. For example, *proposed* and *requested* in a bilateral negotiation cannot both be valid at the same step.

Non-concurrent property for protocol μ:

$$\forall s_0, s_1, \ldots, s_n \in state\text{-}set\ (\mu \vdash (s_0 \leftrightarrow \text{one-of}(\{s_1, \ldots, s_n\}\ )) \wedge \neg(i = j) \rightarrow \neg(s_i \wedge s_j)$$

) where $0 \leq i,j \leq n$

The above property means that if the protocol μ has defined states s_1 to s_n not to have a direct parent, then two or more of these sibling states cannot both be *true*.

6.4.14 Decidable

A decision problem is decidable if it can be solved by an algorithm that halts on all inputs in a finite number of steps. Decidability is valid if there exists a computational

process that solves the problem in a finite number of steps. A protocol, μ, is decidable if it exhibits both strong termination and liveness properties.

Decidable property for protocol μ: μ ⊢ *strong termination* and *liveness* properties.

6.5 Proving the Properties of the Bilateral Protocol

The bilateral protocol, Theory 6.1, is used, as an example to show how the properties of a protocol may be proved or the protocol designed to exhibit certain properties. We analyse and prove whether each of the properties defined in section 6.4 hold in the bilateral protocol. In most of the cases, a proof is provided using structural induction over states and processes.

[{*X* , *Y*}.(*b*:*bilateral-protocol*(¬*negotiating*))] *b*>>*closed*

negotiating ↔ one-of ({ *open*, *closed* }) (1)

open ↔ one-of ({ *requested*, *offered* }) (2)

closed ↔ one-of ({ *agreed*, *rejected*, *timeout* }) (3)

proposed(*X*) → *offered*(*X*) (4)

¬*negotiating* ↔ none-of ({ *open, closed* }) (5)

¬*open* ↔ none-of ({ *requested*, *offered* }) (6)

¬*closed* ↔ none-of ({ *agreed*, *rejected, timeout* }) (7)

¬*negotiating* ↔ [*X.initial_request*]*requested*(*X*) ∨ [*X.initial_offer*]
(*offered*(*X*) ∧ ¬*proposed*(*X*)) ∨ [*X.initial_propose*] *proposed*(*X*) (8)

requested(*X*)↔ [*Y.offer*] (*offered*(*Y*) ∧ ¬*proposed*(*X*)) ∨ [*Y.propose*]
proposed(*Y*) ∨ [*Y.suggest*] *requested*(*Y*) ∧ ¬(*X*= *Y*). (9)

offered(*X*) ↔ [*Y.agree*] *agreed*(*Y*) ∧ ¬(*X*= *Y*). (10)

proposed(*X*)) ↔ [*Y.request*] *requested*(*Y*) ∧ ¬(*X*=*Y*). (11)

open ↔ ([{*X*,*Y*}.*reject*] *rejected* ∨ [*timeout*]*timedout* ∨ [*offered*(*X*))?;
Y.agree] *agreed*(*Y*) **)** ∧ ¬(*X*=*Y*). (12)

Theory 6.1 Logical Theory of Bilateral Protocol

In the bilateral protocol, the set of states, *state-set* = {*negotiating*, *requested*, *offered, proposed, closed, rejected, timedout, agreed, open*}

The set of atomic transitions in the bilateral protocol is *action-set* = {*initial_offer*, *initial_request*, *initial_propose*, *offer*, *request*, *suggest*, *propose*, *agree*, *reject*, *timeout*}

The set of paths, *path-set*, consists of the processes in *action-set* related with ANML connectors through the function *Proc*(*state-set*, *action-set*, {*X*,*Y*}).

6.6 Termination

Termination Property for protocol μ: $\mu \vdash \forall s, s_{pt}$: *state-set* $\forall p_s$:*path-set* $\exists\, p_t$:*path-set* **(**

$$(\textit{terminal-parent}(s_{pt})) \rightarrow (s \rightarrow [p_s ; p_t]\ \ s_{pt}) \textbf{)}$$

where $s_i, s_t \in$ *state-set*

In the bilateral protocol, the state *closed* is a terminal parent state. Termination of the bilateral protocol is proved by showing that from any state all paths lead to the *closed* state. The sub-paths from a given state to *closed* are first derived from the bilateral protocol.

The simplest sub-paths to a successful or unsuccessful *closed* state consists of an atomic transition and are given by {*X*,*Y*}.*exit_paths* = *X.agree* ∪ *X.reject* ∪ *timeout*. The path {*X*,*Y*}.*exit-paths* represents the possible actions leading to the *closed* state in a negotiation between agents *X* and *Y*. Either agent can *agree*, *reject* or a timeout can occur. The sub-path to a *closed* and failed state is given by the process {*X*,*Y*}. *unsuccessful_exit* = *X.reject* ∪ *timeout*. Either agent can *reject* or a timeout can occur. According to the protocol, *unsuccessful_exit* transitions may be chosen at any point in a negotiation. The relation between the process *exit_paths* and the process *unsuccessful_exit* is given by {*X*,*Y*}.*exit_paths* =*X.agree* ∪ *unsuccessful_exit*. The path {*X*,*Y*}.*exit_paths* terminates because it consists of atomic actions leading to the *closed* state.

The logical theory of the bilateral protocol is analysed so as to derive the possible paths to the *closed* state from all of its *open* states. The paths *from_offered*(*X*),

from_proposed(*X*), *from_requested*(*X*) lead to a *closed* state from the *open* states *offered*(*X*), *proposed*(*X*) *and requested*(*X*) respectively.

from_offered(*X*) = *offered*(*X*)?; (*Y.agree* ∪ *X.reject* ∪ *Y.reject* ∪ *timeout*)
= *offered(X)*?; *exit_paths*

from_proposed(*X*) = *proposed*(*X*)?; ((*Y.request*; *from_requested*(*Y*)) ∪ *Y.agree* ∪ *X.reject* ∪ *Y.reject* ∪ *timeout*)
= *proposed*(*X*)?; ((*Y.request*; *from_requested*(*Y*)) ∪ {*X* , *Y*}.*exit_paths*)

from_requested (*X*) = *requested*(*X*)?; ((*Y.offer*; *from_offered*(*Y*)) ∪ (*Y.propose*; *from_proposed*(*Y*)) ∪ (*Y.suggest*;*from_requested*(*Y*)) ∪ *X.reject* ∪ *Y.reject* ∪ *timeout*)

The above 3 paths are the possible paths from an *open* sub-state to the *closed* state. An abstract compositional path can now be deduced for a negotiation process by following an entry to an *open* negotiation with all possible paths to a *closed* state i.e. *negotiation-paths* = *entry-actions*; *paths-to-closed*. There are 3 entry actions as given in the protocol and the *paths-to-closed* are given above. It is left to prove that *negotiation-paths* terminates, thereby proving that all paths lead to the *closed* state and that the bilateral protocol shows the termination property.

negotiation_paths = (*X.initial_offer*; *from_offered*(*X*)) ∪ (*X.initial_propose*(*X*); *from_proposed*(*X*)) ∪ (*X.initial_request*;*from_requested*(*X*))

Each sub-path occurring after the entry actions in *negotiation_paths* is first expanded and simplified for the purpose of expressing *negotiation_paths* in terms of entry and exit paths.

In *from_requested*, the paths *from_offered* and *from_proposed* are replaced with their sub-paths. The process (*X.reject* ∪ *Y.reject* ∪ *timeout*) is equivalent to unsuccessful exiting i.e. *unsuccessful_exit*.

from_requested(*X*) = *requested*(*X*)?; **(**

(*Y.offer*; *offered*(*Y*)?; {*X*,*Y*}.*exit_paths*) ∪
(*Y.propose*; *proposed*(*Y*)?; ((*X.request*; *from_requested*(*X*))∪
{*X*,*Y*}.*exit_paths*)*)* ∪
(*Y.suggest*;*from_requested*(*Y*)) ∪ *unsuccessful_exit* **)**

Assuming all atomic actions terminate and that the corresponding state holds after an atomic action i.e. the process $X.do_{action}$; $done_{action}$? can be abbreviated to $X.do_{action}$. For example, the process *X.propose*; *proposed*(*X*)?; *Y.do_action* is abbreviated to the process *X.propose*;*Y.do_action*. The testing of *proposed*(*X*)? after *X.propose* can be omitted because the test always succeeds by virtue of the preceding atomic action *X.propose* terminating. Hence the tests that are successful, because of the previous atomic action that precedes them, are removed from the process *from_requested*(*X*).

from_requested(*X*) = *requested*(*X*)?; **(**
(*Y.offer*; {*X*, *Y*}.*exit_paths*) ∪
(Y.propose; ((*X.request* ; *from_requested*(*X*)) ∪ {*X*, *Y*}.*exit_paths*) *)* ∪
(*Y.suggest*; *from_requested*(*Y*)) ∪ *unsuccessful_exit* **)**

Distributing *Y.propose* over the union operator in ((*X.request*; *from_requested*(*X*)) ∪ {*X*, *Y*}.*exit_paths*) gives

from_requested(*X*) = *requested*(*X*)?; **(**
(*Y.offer*; {*X*, *Y*}.*exit_paths*) ∪
(*Y.propose*; (*X.request* ; *from_requested*(*X*))) ∪
(*Y.propose*; {*X*, *Y*}.*exit_paths*) ∪
(*Y.suggest*;*from_requested*(*Y*)) ∪ *unsuccessful_exit* **)**

The path *from_requested* is further simplified by distributing and grouping the terms with *exit_paths* and *from_requested* along the set operators. For example
(*Y.offer*;{*X*, *Y*}.*exit_paths*) ∪ (*Y.propose*;{*X*, *Y*}.*exit_paths*) is equivalent to
(*Y.offer* ∪ *Y.propose*); {*X*, *Y*}.*exit_paths*

from_requested(*X*) = *requested*(*X*)?; **(**
((*Y.offer* ∪ *Y.propose*); {*X*, *Y*}.*exit_paths)* ∪

(*(*(*Y.propose*; *X.request*) ∪ *Y.suggest)*; *from_requested*(*Z*))
∪ *unsuccessful_exit*) (*f_r1*)

The path *from_requested*(*X*) has a nested *from_requested* path and is therefore an iterative path set. Currently the path *from_requested*(*X*) is in the form $m = r?; ((a \cup b;m) \cup c)$ i.e. $m = r?;(d \cup b;m) = r?; (d;b^*)$. Using this equation, the termination of *from_requested*(*X*) is explained.

The test *r*? denotes *requested*(*X*)? and terminates since it is just a logical test. The process *a* denotes ((*Y.offer* ∪ *Y.propose*); {*X*, *Y*}.*exit_paths*). The process *a* terminates since both its atomic processes, *Y.offer* and *Y.propose*, terminate and are followed by exit_paths. The process *b* denotes (*Y.propose*; *X.request*) ∪ *Y.suggest*. The process *b* terminates as its 3 sub-processes are atomic. The process *c is unsuccessful_exit* and terminates being exit paths with atomic actions. Thus in the expression $m = r?; ((a \cup bm) \cup c)$, the paths *a*, *b*, *c* and *r* terminate.

The process *m* denotes *from_requested* and is a recursive path equation. Following the earlier definition of termination for a recursive function, the base case is the process *d* - ((*Y.offer* ∪ *Y.propose*); {*X*, *Y*}.*exit_paths*) ∪ *unsucessful_exit*. This base case terminates as all its sub-paths terminate. The path $m=r?; (d;b^*)$ can be solved using algebra and thus all paths in the process *from_requested* lead to a terminal state. The termination property holds for the process *from_requested*.

The process *negotiation_paths* is similarly simplified by substituting for *from_proposed* and *from_offered* with their sub-paths as follows:

negotiation_paths = (*X.initial_offer*; *from_offered*(*X*)) ∪
(*X.initial_propose*; *from_proposed*(*X*)) ∪
(*X.initial_request*;*from_requested*(*X*))

= (*X.initial_offer*; *offered(X)*?; *exit_paths*)
∪ (*X.initial_propose*; *proposed*(*X*)?; *(*(*Y.request*; *from_requested*(*Y*)) ∪
{*X*, *Y*}.*exit_paths)*)
∪ (*X.initial_request*; *from_requested*(*X*))

Grouping terms over *from_requested* and *exit_paths* and removing redundant state testing yields

negotiation_ paths = ((*X.initial_offer* ∪ *X.initial_propose*) ; {*X*, *Y*}.*exit_paths*) ∪
(*(X.initial_request* ∪ (*X.initial_propose*; *Y.request*)*)*; *from_requested*(*Z*)**)**

The termination property has been shown to hold for the path *from_requested*(*X*). All the other sub-paths in *negotiation_paths* terminate being atomic transitions or exit paths. Therefore all paths in *negotiation_paths* lead to *closed* state. **QED.**

The expression *negotiation_paths* is further manipulated to show an interesting representation in terms of entry and exit paths. Replacing the process *from_requested* in *negotiation_paths* with its sub-paths from (*f_r1*) produces

negotiation_ paths = ((*X.initial_offer* ∪ *X.initial_propose*) ; {*X*, *Y*}.*exit_paths*) ∪
((*X.initial_request* ∪ (*X.initial_propose*; *Y.request*)); *requested*(*X*)?; *(*
((*Y.offer* ∪ *Y.propose*); {*X*, *Y*}.*exit_paths*) ∪
(((*Y.propose*; *X.request*) ∪ *Y.suggest*); *from_requested*(*Z*))
∪ {*X*, *Y*}. *unsuccessful_exit)* **)**

Let one entry-point in a negotiation be called *entry-point1* where *entry_point1* = *X.initial_offer* ∪ *X.initial_propose*. Let the second entry-point to a negotiation into a *requested* state be *entry_to_requested* where the path *entry_to_requested* = *X.initial_request* ∪ (*X.initial_propose*; *Y.request*). For clarity let {*X*, *Y*}.*exit_paths* be abbreviated to the path *exit_paths* and the path {*X*,*Y*}.*unsuccessful_exit* be *unsuccessful_exit*. Using these two entry points in a negotiation, removing any redundant terms, testing in the expression *negotiation_paths* and distributing *entry_to_requested* over the sub-paths of *from-requested* gives

negotiation_ paths =
entry_point1; *exit_paths* ∪
(*entry_to_requested*; (*Y.offer* ∪ *Y.propose*) ; *exit_paths*) ∪

(*entry_to_requested* ; ((*Y.propose*; *X.request*) ∪ *Y.suggest*);
from_requested(*Z*)**)** ∪ (*entry_to_requested* ; *unsucessful_exit*)

Let paths to an *offered* state be denoted by *to-offered* where the *offered* state follows a *requested* state. *to-offered* = *entry_to_requested*; (*Y.offer* ∪ *Y.propose*)
Let *to_requested* denote the sequence for entering in a *requested* state, excluding through an entry action, *to_requested* = ((*Y.propose*; *X.request*) ∪ *Y.suggest*).

The process *negotiation_paths* can be further simplified as follows:

negotiation_ paths =
((*entry_point1* ∪ *to-offered*); *exit_paths*) ∪
(*entry_to_requested* ; **(**(*to_requested*; *from_requested*(*Z*)) ∪
*unsuccessful_exit***)**)

In the process *negotiation_ paths*, the path ((*entry_point1* ∪ *to-offered*); *exit_paths*) terminates since ultimately *exit_paths* are followed and since both *entry_point1* and *to-offered*, containing no iteration, terminate. In the path
(*entry_to_requested* ;**(**(*to-requested*; *from_requested*(*Z*)) ∪ *unsucessful_exit***)**)
the sub-paths *entry_to_requested*, *to-requested*, *unsuccessful_exit* all terminate because they do not contain iteration. It has previously been shown all the paths in *from_requested*, which is the recursive part of a negotiation, lead to a terminal state. Hence all the sub-paths in *negotiation_ paths* lead to a terminal (*closed*) state, as shown earlier.

On examining the latest expression for *negotiation_paths*, it can be seen that it consists of direct termination via the *offered* state and a recursive path through the *requested* state. The base case is ((*entry_point1* ∪ *to-offered*); *exit_paths*) showing that a negotiation ultimately passes through the *offered* state for successful exit paths. The recursive case concerns entering and iterating in the *requested* state via the sequential path

(*entry_to_requested*; **(**(*to_requested*; *from_requested*(*Z*)) ∪ *unsuccessful_exit***)**).

Thus the bilateral protocol exhibits the termination property: from all states, all possible paths may lead to a terminal state.

6.7 Strong Termination

Strong Termination Property for protocol μ: $\mu \vdash \forall s, s_{pt}$: *state-set* $\neg\exists\, p_t$:*path-set* **(**

$terminal\text{-}parent(s_{pt}) \wedge (s \rightarrow [p_t^*]\, s_{pt})$**)**

where $s_i, s_t \in$ *state-set*

The strong termination property asserts that there is no iteration in a process. In the path *negotiation_ paths*, where

negotiation_ paths =

((*entry_point1* ∪ *to-offered*); *exit_paths*) ∪

(*entry_to_requested* ; **(**(*to_requested*; *from_requested*(*Z*)) ∪

*unsuccessful_exit***)**)

The sub-process *from_requested* is recursive and therefore the bilateral protocol does not satisfy the strong termination property.

6.8 Liveness

Liveness property for protocol μ: $\mu \models \forall\, s_i$:*state-set*, a_i :*path-set* $(s_0 \leftrightarrow [X_1.a_1]s_1 \vee \ldots. \vee [X_n.a_n]s_n)$

$\rightarrow \neg\exists\, b$: *path-set* (($(Y.b\,?; X_i.a_i) \wedge (X_i.a_i\,?; Y.b)$) ∨ [*timeout*]*Z*)

where $0 \le i \le n$, Y and $X_i \in A\_group$

Liveness, asserted by the *Liveness property*, implies the absence of deadlock in a negotiation where it is never the case that two or more agents wait for each other. The liveness property holds in the bilateral protocol. This is proved by induction by showing that any execution of an allowed path (i.e. process) does not yield a deadlock. It is first proved that there is no deadlock in a base case and then assumed that there is no deadlock in a general state, *s*. We finally prove that all possible atomic actions from state *s* do not induce deadlock. Any new state after *s* contains no deadlock. Thus, it is proved that there is never any deadlock at all states, nor are any caused by possible transitions, atomic or composed. The rules in Theory 6.1 are used for reasoning throughout the proof.

Base Case. A bilateral negotiation is started from a ¬*negotiating* state since the overall parent state is *negotiating* (rule (1) in Theory 6.1). The base case is to prove absence of deadlock in the state ¬*negotiating*. Rule (8) shows that the three actions to start a negotiation are *initial_request*, *initial_offer* and *initial_propose*. Any of these three actions is possible by any of the two agents, *X* and *Y*, since there has been no previous interaction between them. Neither of them has to wait for the other in order to do an entry action. Hence there is no deadlock at ¬*negotiating* for entering a negotiation.

Induction hypothesis. Assuming that there is no deadlock at an arbitrary state, s_0 in the negotiation, any transitions from s_0 to a successor state is possible. If s_0 is a terminal state, then *null* is the next possible action and does not cause deadlock. The possible transitions from s_0 may be given as $s_0 \leftrightarrow [X_1.a_1]s_1 \vee \ldots. \vee [X_n.a_n]s_n$ where a_i is an atomic transition derived from the rules from Theory 6.1.

Induction proof. We prove that performing an action from s_0 does not cause any deadlock in the next state. Using the induction hypothesis, assumptions and proving the absence of deadlocks at any state, s_i, succeeding s_0 after an atomic transition a_i, we will have proved that there is no deadlock at any state in a bilateral negotiation, and thus after any process. All sequences of execution from s_0 are analysed by testing the possible states after each atomic transition from s_0. It is shown that there is no deadlock in all the successor states of s_0.

Induction proof, *Case1*: When s_0 is *offered*(*X*). The possible atomic transition from s_0 is *offered*(*X*) ↔ [*Y.agree*] *agreed*(*X*) ∧ ¬(*X*= *Y*) given by rule (10). From the induction hypothesis, there is no deadlock in *offered*(*X*). *Y* may agree and does not depend on *X*. The *agreed*(*Y*) state contains no deadlock by virtue of being a terminal state and none of the agents need to carry out any actions from there.

Induction proof, *Case2*: When s_0 is *requested*(*X*). The possible atomic transitions are *Y.offer*, *Y.propose* and *Y.suggest*, given in rule (9). By the induction hypothesis there is no deadlock in s_0, the *requested*(*X*) state. The possible actions are for *Y* to make an *offer* or a *propose* or a *suggest* and none of them depend on *X*. It is proved below that

there are still no deadlocks after *Y*'s action resulting in either *offered*(*Y*), *proposed*(*Y*) or *requested*(*Y*).

Induction proof, *Case2a*: s_0 is *requested*(*X*), action is *Y.offer*, next state is *offered*(*Y*). From *offered*(*Y*), *X* can *agree*. *X* waits for *Y* to make an offer to get to *offered*(*Y*) while *Y* waits for *X* to make a request to get to *requested*(*X*) before *Y* offers. The situation in s_0 is the condition *requested*(*X*)?*Y.offer* ∧ *offered*(*Y*)?*X.agree*. Obviously this does not correspond to the condition for deadlock and represents a sequence of offer and agree actions. From the induction hypothesis, there is no deadlock in s_0, *requested*(*X*), and therefore *Y* can make an offer. All atomic transitions terminate rendering *offered*(*Y*) eventually *true* if *Y* makes an offer. *X* can then agree. There is no deadlock in *offered*(*Y*).

Induction proof, *Case2b*: s_0 is *requested*(*X*), action is *Y.propose*, next state is *proposed*(*Y*). From *proposed*(*Y*), *X* can request or agree. The formula *requested*(*X*)? *Y.propose* ∧ *proposed*(*Y*)? (*X.agree* ∪ *X.request*) holds at s_0. As before, there is no deadlock in *requested*(*X*) such that *Y* can make a propose which terminate. Therefore state *proposed*(*Y*) comes to pass. *X* can then request or agree. The state *proposed*(*Y*) is not deadlocked.

Induction proof, *Case2c*: s_0 is *requested*(*X*), action is *Y.suggest*, next state is *requested*(*Y*). From *requested*(*Y*), *X* can offer, propose or suggest. The formula *requested*(*X*)? *Y.suggest* ∧ *requested*(*Y*)? (*X.offer* ∪ *X.propose* ∪ *X.suggest*) holds at s_0. For the same reasons as above, *requested*(*Y*) becomes true. After that, *X* does not depend on *Y* to make an *offer*, *propose* or *suggest*. Thus *requested*(*Y*) contains no deadlocks.

Induction proof, *Case3*: When s_0 is *proposed*(*X*). The possible actions from s_0 are *Y.request* and *Y.agree*. In case of the action *agree*, the condition, *proposed*(*X*)?*Y.agree*, is true at s_0 leading to *agreed* as successor state. By the induction hypothesis there is no deadlock in *proposed*(*X*) and *Y.agree* terminates. Since *agreed* is a terminal state, it has no deadlocks. If *Y* chooses to make a request then *proposed*(*X*)?*Y.request* ∧ *requested*(*Y*)?(*X.offer* ∪ *X.propose* ∪ *X.suggest*). By the same arguments as before, *requested*(*Y*) becomes true and contains no deadlocks.

Induction proof, *Case4*: When the state s_0 is *open*. The possible transitions are given in rule (12): *open* ↔ ([*X.reject*] *rejected* ∨ [*timeout*] *timedout* ∨ [*offered*(*X*)?*Y.agree*] *agreed*(*Y*) **)** ∧ ¬(*X*=*Y*). There is no deadlock in the *open* state (Ind. Hypothesis) and atomic transitions terminate rendering *rejected* and *timedout* possible. Neither agent waits for the other to do a *reject* and a *timeout* event depends only on the time elapsed. The states *rejected* and *timedout* are terminal states with no deadlock. The test condition is now considered.

The interesting feature for this property occurs when there is a test condition in a rule such as the test *offered*(*X*)?. This test checks whether agent *X* has made an offer. Agent *X* makes an offer only when agent *Y* has enabled the *requested*(*Y*) state through a *request* or an *initial_request*. Consider only *Y.request* for simplicity and since both cases are similar. This yields in s_0 the condition

(*offered*(*X*)?;*Y.agree*)∧(*requested*(*Y*)?;*X.offer*), that is
(*X.offer*?;*Y.agree*)∧(*Y.request*?;*X.offer*).

This condition does not unify with the condition for deadlock, (*X.action2*?; *Y.action1*) ∧ (*Y.action1*?; *X.action2*). There is no deadlock since although agent *X* has to make an offer before agent *Y* can agree, *X* is not waiting for *Y* to agree before *X* can make its offer. In fact agent *X* is waiting for agent *Y* to do another action i.e. a *request*. In order to make a request agent *Y* does not depend on *X* making an offer or itself to agree - *Y.request* is not possible from an *offered* or *agreed* state.

In this sequence of atomic actions, two transitions do not depend mutually on each other but rather occur sequentially. As shown before there is no deadlock in s_0, *requested*(*Y*), thus the process *Y.request*?;*X.offer* eventually happens enabling the process *X.offer*?;*Y.agree*. The testing of *offered*(*X*) and the atomic action *agree* terminates rendering *agreed*(*Y*) true. We have proved that *X.offer*?;*Y.agree* can occur and it causes no deadlock in the terminal *agreed* state. Therefore there are no deadlocks in the *open* state by the induction hypothesis and neither in its successor states, being terminal states.

Induction proof, *Case5*: When s_0 is *closed*, *rejected, timedout or agreed*(X). This is a trivial case. These four states being terminal states, there are no transitions possible from them. Deadlock is absent.

Induction proof, *Case6*: When the state s_0 is ¬*negotiating*. The three possible entry actions are *initial_request*, *initial_offer*, *initial_propose*. The induction hypothesis (and base case) asserts that there is no deadlock at ¬*negotiation* and agent X can perform one of these actions without waiting for agent Y as these are entry actions. All the transitions are atomic and terminates making states *requested*(X), (*offered*(X)∧¬*proposed*(X)) or *proposed*(X) eventually true. None of these states contain a deadlock as there are no actions possible before ¬*negotiating*. Agent Y does not need to perform any action before X can make these entry actions.

Induction proof, Case7: When s_0 is negotiating.
This state is dealt by analysing all of its sub-states in the above cases.

We have proved that for all possible states s_0, all transitions from an arbitrary state s_0 maintain liveness. **QED.** The bilateral protocol satisfies the liveness property.

6.9 Soundness

Soundness property for protocol μ: $\forall A: \varepsilon \ ((\mu \vdash A) \rightarrow (\mu \models A))$

The soundness property between a protocol, μ, and a process, δ, which is an instance of μ can be clarified as below:

$$\mu \vdash (s_0 \leftrightarrow ([a_1]s_1 \vee \ldots \vee [a_n]s_n)) \rightarrow \neg(\delta \vdash (s_0 \rightarrow ([a_z]s_z))$$

where $\neg(0 \leq z \leq n)$

for all states s_0 to $s_n \in$ *state-set*, a_1 to $a_n \in$ *path-set*, s_z: ε and a_z: Ψ

All states, processes and transitions occurring in a negotiation instance δ, are only those defined by the sound protocol, μ. If the protocol μ is sound, the process δ does not yield any unexpected states. This property ensures the validity of negotiation paths and states, e.g. an *agree* action cannot directly follow a *request* action nor can a

state be both *agreed* and *rejected*. There are no erroneous states or transitions. The soundness property holding for the bilateral protocol is proved by induction by showing that all possible transitions from any state preserve soundness.

The base case proves that the protocol is sound after one atomic transition. This case is covered by the logical theory of the protocol. The induction hypothesis assumes that the soundness property holds at an arbitrary state s_1 after performing the path p_d, where $p_d \in$ *path-set* and may be a *null* process. We then prove the negotiation remains sound when process p_d is either followed or preceded by any valid atomic action (that is an element of *action-set*). This implies a negotiation instance is sound after the execution of any paths.

Base Case. The base case for proving the bilateral protocol is sound is given by the action-condition rules in Theory 6.1. Atomic transitions in *action-set* lead from a well-defined source state to a well-defined target state, both being in *state-set*. The valid states and atomic transitions in *state-set* and *action-set* respectively can be inferred from the protocol. From the ¬*negotiating* state, entry transitions lead to an *open* state and are given by rule (8) in Theory 6.1. Likewise atomic transitions from *requested, offered, proposed* and *open* are given by rules (9) to (12). Rules (1) to (7) on the relation between states and sub-states ensure that the parent states of the current sub-state are *true* and all other states are *false* as required.

For all atomic transitions p: *action-set,* there is a source state s_1: *state-set* and a well-defined target state s_2:*state-set*. If p is equivalent to *null* then s_1 and s_2 are the same state e.g. *closed* states are terminal states with no transitions. Theory 6.1 gives all the possible atomic transitions.

Induction hypothesis. For an arbitrary state s_n: *state-set* there is a path p_d: *path-set* that yields expected state s_d: *state-set* given by both the protocol μ and the negotiation instance δ. ($\mu \vdash s_n \rightarrow [p_d]s_d$)).

Induction Proof. It is proved that soundness is maintained when the path p_d is followed or preceded by an atomic transition, *a*:*action-set* i.e. the path (p_d;*a*) or (*a*; p_d) preserves soundness.

Induction Proof, *Case1*. We first prove soundness for the case (p_d;*a*) by showing that the transition $s_n \rightarrow [p_d ; a]\ s_{d+1}$ yields a well-defined state s_{d+1}. From the composition axiom in chapter 3, a complex path can be decomposed into the execution of its sub-processes with an intermediate state holding after each sub-process.

$$[\alpha; \beta]\ state_\beta \leftrightarrow ([\alpha] state_\alpha \wedge (state_\alpha \rightarrow [\beta]\ state_\beta)) \leftrightarrow ([\alpha]\ state_\alpha?; [\beta]\ state_\beta) \leftrightarrow [\alpha][\beta]\ state_\beta \text{ where } state_\beta, state_\alpha \in \varepsilon$$

The formula $s_n \leftrightarrow [p_d ; a]\ s_{d+1}$ is equivalent to $s_n \leftrightarrow [p_d][a]\ s_{d+1}$
This means that there is an intermediary state, s_i: *state-set*, between executing the path p_d and the atomic transition *a*.

$$s_n \rightarrow [p_d]\ s_i?; [a]\ s_{d+1}$$
$$s_n \rightarrow [p_d]\ s_i \wedge s_i \leftrightarrow [a]\ s_{d+1}$$

From the induction hypothesis, the formula $s_n \rightarrow [p_d]\ s_d$ gives a sound negotiation and a well-defined state s_d. From the logical theory and the base case, the atomic transition $s_i \rightarrow [a]\ s_{d+1}$ preserves soundness. The state s_d is made equivalent to s_i i.e. $s_d \leftrightarrow s_i$. The state s_d is known to be well-defined from the induction hypothesis and therefore so is s_i, which is thus in *state-set*. The resulting two transitions are $s_n \rightarrow [p_d]$ s_d and $s_d \rightarrow [a]\ s_{d+1}$. Since s_d and s_i are in *state-set* and action *a* is in *action-set*, then s_{d+1} is in *state-set* given by corresponding action-condition rules in the protocol. By the induction hypothesis and the logical theory, both transitions yield a sound system.

The path (p_d ; *a*) has s_d as the intermediary state. Using Theory 6.1, s_d may be instantiated to an appropriate intermediate state by analysing which state after path p_d is the pre-condition to the execution of *a*. The transition $s_n \rightarrow [p_d ; a]\ s_{d+1}$ preserves the soundness of the negotiation since its sub-paths p_d and *a* do so. A negotiation in the state s_n is sound and remains sound at s_{d+1} after an atomic transition. Having

proved that an atomic transition in *state-set* preserves the soundness of a negotiation, and assuming that a path p_d preserves soundness, we have proved that the path consisting of p_d ;a also yields a sound negotiation.

Induction proof, *Case2*. The proof for soundness when preceding the path p_d with an action is similar. It is proved that $s_{n-1} \rightarrow [a; p_d]\ s_d$ maintains a sound negotiation for *a*:*action-set*. From the base case, any atomic transition in *action-set* provides a sound negotiation. The induction hypothesis asserts that a path p_d preserves the soundness property where $s_n \rightarrow [p_d]\ s_d$ gives a well-defined state s_d. We need to prove that the path a; p_d also yields a sound negotiation at all states.

By the definition of process composition in ANML, $s_{n-1} \rightarrow [a; p_d]\ s_d$ is equivalent to $s_{n-1} \rightarrow [a]\ [p_d]\ s_d$ where there is an intermediate state, s_i, after executing atomic transition a and before executing the process p_d.

$$s_{n-1} \rightarrow [a]\ s_i?;\ [p_d]\ s_d$$
$$s_{n-1} \rightarrow [a]\ s_i \wedge s_i \leftrightarrow [p_d]\ s_d$$

The state s_n is made equivalent to the intermediary state s_i making thus s_i be in *state-set*. The action, a, is in *action-set*, therefore from the logical theory and the base case, s_{n-1} is in *state-set* and is well-defined. Since the path p_d and the action a preserves soundness giving s_{n-1} and s_d to be in *state-set*, the path a; p_d preserves the soundness property as its sub-paths do so. **QED**

Having proved that an atomic transition gives a sound negotiation, and assuming that a path p_d preserves soundness, it has been shown that the paths a; p_d and p_d; a maintain soundness. By induction, all paths in a bilateral negotiation will preserve the soundness property. The theory of the bilateral protocol is sound.

6.10 Serial

serial property for protocol μ: $\mu \vdash \forall\ s_i$:*state-set*, a_i :*path-set* $(s_0 \rightarrow [a_1;\ldots;\ a_n]s_n)$

$$\rightarrow \neg(\ s_i \rightarrow [p_i]\ s_j \wedge\ [p_k]s_k\ )$$

where $(0 \leq i, j, k \leq n)$

for all states $s_0, s_1, s_p, s_q \in$ *state-set*, and a_1 to a_n , p_1 to $p_n \in$ *path-set*.

A protocol is serial if all its derivable paths are executed sequentially and there are no parallel executions.

The rules in Theory 6.1 are used to analyse whether the bilateral protocol is serial. The possible actions are analysed from all states in the bilateral protocol. According to rule (12) in Theory 6.1, a *reject* or *timeout* event can occur from an *open* state leading to a *closed* state, but these two actions cannot occur in parallel with each other nor with other transitions as given by the bilateral protocol. From rule (3), the state *rejected* and the state *timedout* cannot both be *true* at the same instant. We analyse below whether any parallel actions are possible from all non-terminal states in a bilateral negotiation.

- *from* ¬*negotiating*

  The three actions possible from the state ¬*negotiating* are *initial_request*, *initial_offer* and *initial_propose* leading to the states *requested, offered* and *proposed* respectively. From rules (2) and (6), an *initial_request* cannot be in parallel with an *initial_offer* as the states they trigger cannot co-exist. From rule (8), by the condition (*offered*(*X*) ∧ ¬ *proposed*(*X*)), an *initial_offer* cannot be executed in parallel with an *initial_propose*. Hence, no parallel actions are possible from the state ¬*negotiating*.

- *from offered*(*X*)

  From rules (10) and (12), only an agreement, a rejection or a timeout can occur from this state. From rules, (3) and (7), the states *agreed*, *rejected* and *timedout* cannot co-exist. No parallel actions are possible from the state *offered*(*X*).

- *from requested*(*X*)

  From rule (9), the successor states may be either *offered*(*Y*), *requested*(*Y*) or *proposed*(*Y*). From rules (2) and (6), the states *offered*(*Y*) and *requested*(*Y*) cannot both be *true*. From rule (9), the condition (*offered*(*X*) ∧ ¬ *proposed*(*X*)

do not allow these two states to co-exist. Therefore no parallel actions are possible from *requested*(*X*).

- *from proposed*(*X*)

  The state *proposed*(*X*) is a sub-state of *offered*(*X*) and so has the same possible transitions as from *offered*(*X*) with in addition the action *Y.request*. By rule (10), the actions from *offered*(*X*) lead to a *closed* state. The action *Y.request* leads to an *open* state. From rule (1), the states *open* and *closed* cannot both be true.

Therefore there are no parallel processes according to a bilateral protocol and the actions are performed sequentially. The bilateral protocol is serial.

6.11 Ordered

The ordered property is respected if none of the sequential processes can occur in parallel and there is not a different order of actions leading to the same state.

Ordered property for protocol μ: $\mu \vdash \forall\ s_i$:*state-set*, a_i :*path-set* $((s_0 \rightarrow [a_1;\ldots; a_n]s_1)$

$\rightarrow \neg\ (s_0 \rightarrow [a_i; \ldots; a_j]\ s_1)$

where $(0 \leq i,j \leq n)$ and $i\ ..j$ is not equivalent to $1..n$

for all states s_0 to $s_1 \in$ *state-set*, and a_1 to $a_n \in$ *path-set*.

Counterexample from the two following possible paths with the same source and final states:

requested(*X*) ↔ [*Y.propose*; *X.request*; *Y.suggest*; *X.suggest*] *requested*(*X*)

requested(*X*) ↔ [*Y.suggest*; *X.suggest*; *Y.propose*; *X.request*] *requested*(*X*)

The bilateral protocol is not ordered.

6.12 Not Serialisable

A path $(a_1;\ldots; a_n)$ is serialisable iff there exists another path equivalent to it but with $(a_1;\ldots; a_n)$ in a different order.

Serialisability property for protocol μ: $\mu \vdash \forall\ s_i$:*state-set*, a_i :*path-set* $((s_0 \rightarrow [a_1;\ldots;$

$a_n]s_1) \leftrightarrow (s_0 \rightarrow [a_i\,;\ldots;\, a_j]\ s_1)$

where $(0 \leq i,j \leq n)$ and at least one of the numbers in $i..j$ is not equivalent to its corresponding number in $1..n$

for all states s_0 to $s_1 \in$ state-set, and a_1 to $a_n \in$ path-set.

Consider an arbitrary state s_0 in the bilateral protocol and a general transition from state s_0, consisting of two sub-processes as in $s_0 \rightarrow [p_i\,;\, q_j]\ r_j$. There is an intermediate state between processes p_i and q_j.

$s_0 \rightarrow [p_i]\ s_i \wedge s_i \rightarrow [q_j]\ r_j$

The formula $s_0 \rightarrow [p_i\ ;\ q_j]\ r_j$ can be inferred from the following two rules in the protocol:

$s_0 \leftrightarrow [p_1]\ s_1\ \vee \ldots \vee [p_i]\ s_i\ \ldots \vee [p_m]\ s_m$

$s_i \leftrightarrow [\,q_1]\ r_1 \vee \ldots \vee [q_j]\ r_j\ \ldots.\ \vee [q_n]r_n$

where s_x , r_x : state-set and p_x and q_x : path-set

But the re-ordering of the processes, p_i and q_j, giving formula $s_0 \leftrightarrow [q_j\,;\, p_i]\ r_j$ may not hold in a protocol because:

1. The process q_j may not be possible from state s_0.
2. The sequential process (q_j; p_i) may not be possible because none of the resulting states, r_j, from process q_j is a source state for process p_i. The state r_j is not equivalent to any states, s_i, from s_1 *to* s_m.
3. The process p_i does not trigger the state r_j.

Counter-example to show bilateral protocol is not serialisable:
From the ¬*negotiating* state, using rules (8) and (9) the path
¬*negotiating* → [*X.initial_request*; *Y.offer*] *offered*(*Y*) can be derived.
The intermediate state is *requested*(*X*). It is not possible to re-order this path to make an *offer* before an *initial_request*. The formula ¬*negotiating* → [*X.offer*; *Y. initial_request*] *offered*(*Y*) does not hold in the bilateral protocol.

1. The double implication in rule (8), the action *Y.offer* is not valid from the state ¬*negotiating*.
2. The action *X.initial_request* cannot follow the action *Y.offer*.
3. The action *X.initial_request* does not trigger the state *offered*(*Y*).

Therefore, the bilateral negotiation protocol is not serialisable.

6.13 Complexity

If there are p loops with no nesting in a protocol and these iterations are performed n times, then the complexity of the protocol is $p \times n$, that is of order n^2. The complexity of protocols with nested loops is of the order n^n. Nesting loops are loops that contain loops and have increased complexity.

There are 2 iterations in the bilateral protocol: in the *requested* state through a *suggest* action and between the *requested* and *proposed* states via a sequence of *propose* and *request* actions.

The shortest path in a bilateral negotiation consists of 2 atomic actions, as in the formula $\neg negotiating \rightarrow [X.initial\_\ offer;\ Y.agree]\ agreed(Y)$

The worst case entails entering the negotiation and performing n times the action *suggest* and m times the sequence (*propose*; *request*). The worst case of a bilateral negotiation gives rise to the process $(propose;\ request\ ;suggest^n)^m$, followed by a transition to the *offered* state and a terminal action to a *closed* state. The worst case complexity of the bilateral protocol is thus $n \times m$ actions, that is of the order n^2.

6.14 Security

Security Property for protocol μ: $\neg\ \exists\ s_i, s_j \in state\text{-}set, \forall\ p_e \in path\text{-}set$

$$(\mu \vdash (\ entry(p_e) \wedge \neg outermost(s_i) \wedge s_i \rightarrow [p_e]\ s_j\ ))$$

The security property asserts that it is not possible to execute the entry process p_e from a state s_i, where s_i is not the outermost parent state.

From rule (1), the outermost state in the bilateral protocol is *negotiating*. From rule (8), the only entry actions in the bilateral protocol that are possible from ¬*negotiating* are:

The set of entry actions = *entry-actions* ={*initial_request*, *initial_propose*, *initial_offer*}

From the bilateral protocol the actions that are possible from *negotiating* are: *open-actions* ={*request, offer, suggest, propose, timeout, reject, agree*}. These are the actions possible once the negotiation has started.

The set *entry-actions* ∩ *open-actions* = {}. Therefore, the bilateral protocol is secure. There is no way for an agent to restart the negotiation once inside it, ensuring this aspect of security.

6.15 Non-Consecutive Property

Non-consecutive property for protocol μ: $\forall s, s_0, s_1 \in$ *state-set*, $p, p_0 \in$ *path-set* ($\mu \vdash$

$(s_0 \rightarrow [Y.p_0]\, s\,) \wedge (s \rightarrow [X.p]s_1\,) \rightarrow \neg (X = Y)$)

The non-consecutive property never allows an agent to perform two consecutive actions without a transition from another agent. Parameterisation of actions and states with agents ensures that the processes are executed in the correct sequence.

Counterexample for bilateral protocol: An agent can *reject* after it has sent a *request*. Therefore the bilateral protocol does not satisfy the non-consecutive property.

6.16 Reliability

A reliable protocol entails that a negotiation process γ will eventually achieve its goal. In a reliable protocol, there is a possible path to a goal state *g*, providing reachability of goal. Let the predicate *goal*(*g*) return *true* if state *g* is a goal state.

Reliability Property for protocol μ: $\forall s,g$: *state-set*($\mu \vdash \exists p$:*path-set* (*outermost*(*s*) ∧

$goal(g) \rightarrow (\neg s \rightarrow <p>g)))$

The bilateral protocol is reliable if the property $\neg negotiating \rightarrow \langle p \rangle g$ holds. It is possible to reach any state $\in$ *state-set* from the state $\neg$*negotiating*.

A goal state from the bilateral protocol is an element of the set *state-set*.

state-set = {*negotiating*, *requested*, *offered, proposed, closed, rejected, timedout, agreed, open*}

We analyse whether each of these states are possible from the $\neg$*negotiating* state.

From rule (8) in Theory 6.1, the states *negotiating*, *open*, *requested*, *offered, proposed* are reachable from the entry actions. From rules (1) and (12), the states *closed, rejected, timedout* and *agreed* are possible through a sequence of an entry action followed by an exit action. Hence, the reliability property holds in the bilateral protocol.

6.17 Consistency

A protocol is consistent when for all states *A,* it is never the case that $(A \wedge \neg A)$. A consistent protocol contains no contradiction in states.

Consistency property for protocol μ: $\forall\, s \in$ *state-set* $(\neg(\mu \vdash (s \wedge \neg s))$.

The consistency of the bilateral protocol is proved by induction. For all states $s \in$ *state-set*, $\neg(\mu \vdash (s \wedge \neg s))$ is proved to hold for the bilateral protocol μ. The proof is achieved by first proving that starting a negotiation does not produce inconsistency. Then, assuming that there is no inconsistency at state *s*, it is proved that executing an atomic action, *a*, maintains consistency.

Base case. In a bilateral protocol, performing the entry actions from the $\neg$*negotiating* state maintains consistency. From rules (1) to (7) in Theory 6.1, all non-current states are *false*. The protocol is shown to be consistent in the $\neg$*negotiating* state.

The entry actions are *X.initial-request*, *X.initial-propose* and *X.initial-offer*, triggering the *requested*(*X*), (*offered*(*X*) $\wedge$ $\neg$*proposed*(*X*)) and *proposed*(*X*) states respectively.

Rules (1)-(7) implies that when state *requested*(*X*) is *true*, all other states except *open* and *negotiating* are *false*. There is no inconsistency in the *requested*(*X*) state. By the same argument, consistency is maintained for the states *proposed*(*X*) and *offered*(*X*).

Induction Hypothesis. Assuming that a bilateral negotiation is consistent at state

s_0 :*state-set*.

$\mu \vdash \neg(s_0 \wedge \neg s_0)$

Induction Proof. We prove that consistency is still preserved after atomic transition, *a*, from state s_0. In other words we prove the transition $(s_0 \rightarrow [a]\ s_1)$ does not produce any inconsistency in state s_1, by proving from the bilateral protocol that consistency is maintained after each atomic transition, *a*, from each state s_0.

- s_0 is ¬*negotiating*
  Consistency after atomic transitions from ¬*negotiating* is already proved in the base case.

- s_0 is *requested*(*X*)
  From rules (9) and (12) in Theory 6.1, the atomic transitions from s_0 are *Y.offer, Y.propose, Y.suggest*, *timeout X.reject* and *Y.reject*. The process *Y.offer* validates the state (*offered*(*Y*) ∧ ¬*proposed*(*X*)) and the axioms (1) – (7) between states prove that only *offered*(*X*), *open* and *negotiating* are *true*. The rest of the states are *false*. All the resulting states after *Y.offer* are consistent. Similarly from Theory 6.1, the atomic transitions *Y.propose* and *Y.suggest* preserve consistency. The terminal atomic transitions *reject* and *timeout* can also be proved to lead to consistent terminal states.

- s_0 is *offered*(*X*)
  The atomic transition *Y.agree* triggers the state *agreed*(*Y*) from s_0. Axioms (1) to (7) in Theory 6.1 allow only the states *agreed*(*Y*), *negotiating* and *closed* to be *true* and the rest *false*. The actions *reject* and *timeout* lead to consistent terminal states.

- s_0 is *proposed*(X)

  From s_0, either the atomic transition *Y.agree* triggers the state *agreed*(Y) or the action *Y.requested* leads to the state *requested*(Y). The states resulting from both actions can be shown to be consistent from the bilateral protocol.

- s_0 is *open*

  The possible atomic transitions are ($X \cup Y$).*reject* to *rejected* state or *timeout* to *timedout* state or if *offered*(X) then *Y.agree* to *agreed*(Y) state. These are terminal states and the rules for the relation between *closed* and *open* state ensure consistency.

Therefore in the transition ($s_0 \rightarrow [a]\ s_1$), the bilateral protocol does not produce any inconsistency in any state s_1 from each possible state s_0. The consistency property holds for the bilateral protocol.

6.18 A Discussion about Equivalency between Protocols

"Informally, two programs are equivalent when no observations can distinguish them. Further, two subprograms or program phrases are congruent if the result of placing each of them in any program context yields two equivalent programs" [Hennessy and Milner 1985]. Two protocols may be equivalent and both be used as the common protocol in a negotiation. Each agent in a group may have its own individual protocol, which is semantically and syntactically equivalent to the protocol that the group is following. For example consider an English auction between a Dutch agent and a French agent. Both agents have the same understanding of an English auction but their representation of its protocol differs. It can be established that their individual protocols are equivalent so that the agents can privately use their own protocols. The Dutch agent does not need to learn or understand French and vice versa.

6.18.1 Types of Equivalences between Protocols

A number of protocols can be proposed for interactions and equivalence between these protocols ensures that they can be used for the same type of interactions. One solution would be to assess various protocols against a particular set of properties.

Various protocols can be providing different functions or having equivalent locutions with different names or conform to some specification that may be proposed as a standard by an electronic institution.

[McBurney and al. 2002] analyse similarity between protocols according to different types of equivalences. This section adapts their work on dialogue games to interaction protocols (given it is one of the rare papers in dealing with similarity between agent dialogues). Dialogue games are different from interaction protocols which are explicitly specified. In order to analyse the equivalence between interaction protocols, an interaction protocol can be broken into various parts, mostly rules:

- Commencement rules for starting a negotiation.
- Locutions to indicate permitted speech-acts or messages e.g. *propose*, *offer*, *agree*.
- Sequencing rules to define the context under which particular locutions are permitted or obligatory or not. For example, only *offer* or *propose* after a *request*.
- Commitments are rules that define the circumstances under which participants commit to a negotiation e.g. *agree* to an *offer* or even make an *offer*.
- Termination rules that define the circumstances under which the negotiation ends.

Given the above, interaction protocols may be classified according to the group's or agents' goals – information-seeking, inquiry when participants collaborate for seeking an answer, persuasion, negotiation involving bargaining or deliberative for determining a course of action [Walton and Krabbe 1995]. Protocol instances can include a mixture of the different types of protocols primitives. In addition protocols can be classified under which simplest form of protocol they extend. For example, the bilateral protocol may at its simplest only consist of alternative offers, then in the next step move towards richer protocols that include proposals and counter-proposals. After that, the bilateral protocol can be extended to Theory 6.1 which in turn can be extended to richer forms included the one in chapter 4 called the Expanded Bilateral

protocol. All the different ramifications of the bilateral protocol have some parts that are similar and can be classified under the family of bilateral negotiation protocols.

From the above classifications, various types of equivalence are possible between agent interaction protocols adapted from [McBurney and al. 2002]:

- Syntactic equivalence
  Two protocols are syntactically equivalent if their locutions, commencement, combination, commitment and termination rules are the same for both protocols. That is the syntax of the two protocols are the same. Protocols with different names but with same semantics are classified as different in syntactic equivalence.

- Bisimulation equivalence
  This equivalence holds if any state transition achievable in one protocol is also achievable in the other protocol.

- Operation equivalence
  Two protocols are operationally equivalent if for any terminating interaction path in one protocol, there is a similar terminating path in the other protocol and vice versa. This equivalence ignores the length of the paths and therefore the occurrence of iterative actions.

- Equal-length operational equivalence
  Two protocols are equal-length operational equivalent if for any terminating path of actions under one protocol, the same path with the same number of actions terminates the other protocol and conversely.

- Similar-length operational equivalence
  Two protocols are similar-length operational equivalent if for any terminating path of actions in one protocol, there is a similar terminating paths of actions in the other protocol with approximately the same number of utterances and conversely.

[McBurney and al. 2002] also define *T*-similar operational equivalence between two protocols with the same paths consisting of the same subsets of paths. They further analyse the relationships between the various types of equivalence. They propose that syntactic equivalence is a subset of bismulation equivalence and equal-length operational equivalence is a subset of *T*-similar operational equivalence which is a subset of operational equivalence. They also prove that bismulation equivalence is a subset or equal to equal-length operational equivalence. However it is possible for protocols to be equal-length operational equivalent but not bisimulation equivalent. Finally relations between the equivalence types are ordered by set inclusion where syntactic equivalence is a subset of bisimulation equivalence which is a subset of equal-length operational equivalence which is a subset of *T*-similar operational equivalence which is a subset of operational equivalence.

6.18.2 Related Work on Equivalence in Petri Nets

In recent years there has been much work on determining when two nets can be said to be equivalent. A number of equivalences based on bisimulations have been proposed [Reisig 1985]. [Haar and al. 2000] analyse equivalence and more specifically observational equivalence in timed state automata and timed Petri nets and define a translation between the two notations.

[Reisig 1985] calls two conditions and events nets equivalent if their places and transitions correspond to each other according to isomorphism and bijective relations. Equivalent Petri nets have the same number of events, places and steps. If two Petri nets are equivalent, then if the liveness and the cyclic properties hold for one Petri net, it also holds for the other. Two condition events systems (Petri nets) are said to be equivalent if and only if their case graphs are isomorphic [Reisig 1985].

When comparing semantic equivalences for concurrency, it is common practice to distinguish between linear time and branching time equivalences [Pnueli 1985]. Behaviour equivalence is to basically test if two concurrent system specifications as Petri nets have the same behaviour. Bisimulation is a fundamental notion that characterizes behavioural equivalence of concurrent systems. There has been a lot of

research on efficient algorithms for bisimulation checking e.g. can be based on structural reduction rules. There are different types of equivalence for Petri nets:

- Trace equivalence
  The standard example of a linear time equivalence is trace equivalence as employed in [Hoare 1980]. This equivalence is based on analysing the possible runs from Petri nets. Trace equivalence utilises linear time to describe process algebra.
- Bisimulation equivalence
  Bisimulation equivalence relation [Milner 1989], is defined in the context of process algebras, and is a finer equivalence relation than trace equivalence and distinguishes states based on branching properties. Bisimulation equivalence utilises branching time.
- Observational equivalence
  The standard example of a branching time equivalence is observation equivalence or bisimulation equivalence as defined by [Milner 1980]. Observational equivalence is also called weak bisimulation equivalence and is a refined notion of behavioural equivalence. Observation equivalence can be too coarse in its identifications as illustrated by the problems that it may cause in practical applications and analysis [Graf and Sifakis 1987]. See [Hennessy and Milner 1985] for more on observational equivalence.
- Process equivalence
  [Cherkasova and Kotov 1989] introduce the notion of process equivalence and its axiomatisation for process algebra.
- Semantic equivalence

6.18.3 Equality Problem in Petri Nets

Determining the various equivalences becomes undecidable for the strong versions of the equivalences for Petri nets (when all the labels carried by the transitions of the net are assumed to be visible actions). One of the features in process algebra is that of abstraction, providing with a mechanism to hide actions that are not observable, or not interesting for any other reason. By abstraction, some of the actions in a process are made invisible or silent. Consequently, any consecutive execution of hidden steps

cannot be recognized since they are not observed. The weak version of the equivalence (which can be decidable) allows some transitions to be hidden and their firing to be unobservable. [Jancar 1995] demonstrates high undecidability of weak bisimilarity for Petri Nets.

The equality problem relates to showing that two different marked Petri nets with the same number of transitions (but perhaps different numbers of places) will generate the same sequence of transition firings or that two different marked Petri nets with the same number of places (but perhaps different numbers of transitions) will generate the same reachability set. This might allow to modify Petri nets to increase parallelism, decrease the cost of implementation, or other optimisations (which can themselves be intractable).

These cases are concerned with determining if two Petri nets are equivalent or if one is a subset of the other. These problems require to define the notion of equivalence or containment carefully. If equivalence is defined as equal reachability sets, then the number of places cannot be changed. On the other hand, if equality of sets of transition firing sequences is required, transitions may not be changeable. The definition of the problem is therefore quite important. The equality problem that may arise is undecidable i.e. there is no algorithm for determining if two sets of possible firing sequences from an initial place are equal for any two Petri nets *N* and *N'*.

6.18.4 Equivalence in Protocols using ANML

There is a phase, before a negotiation, where the agents establish what protocol they are all going to comply with. If the agents can determine that their individual protocols are equivalent, this removes the onus of constructing a common protocol of negotiation and making all the participants understand it. Learning a new protocol would involve learning about issues such as language, culture and ontology. Establishing equivalency between protocols encourages reuse, thereby reducing the costs and misunderstandings with respect to new protocols. Representing a protocol as a theory in the ANML meta-language allows using rules of logical inference and deduction to prove equivalency between the theories. Such equivalence between different protocols may be proved syntactically and semantically.

Two protocols with logical theories p_1 and p_2 are syntactically equivalent iff $p_1 \vdash p_2$ and $p_2 \vdash p_1$. Two protocols with logical theories p_1 and p_2 are semantically equivalent iff $p_1 \models p_2$ and $p_2 \models p_1$. Both equivalences include checking that redundant rules or locutions in one protocol still detect it as similar to another protocol that does not consist of redundant rules.

A semantic proof pertains to an ongoing problem in agent communication about obtaining common ontologies between agents. Regarding the scenario between the Dutch and the French agents, the agents need to relate Dutch to French messages and French to Dutch according to the context, rendering this a matter of the semantics of a protocol and of translating between languages. Semantic equivalency between protocols is not proved in this section and leads to an interesting avenue for further work.

The syntactic equivalence between two protocols represented in ANML can be established. *Protocol*$_1$ is syntactically equivalent to *Protocol*$_2$ if their logical theories are syntactically equivalent. This is one of the advantages of expressing a protocol as a theory, so that its properties can be logically verified.

A check for syntactic equivalence may be automated through a Prolog program. Let the theory of a protocol be represented as a list of formulae i.e. action condition rules. The predicate *same_protocol* returns *true* if every rule from *Protocol*$_1$ can be derived from *Protocol*$_2$ and vice versa.

same_protocol(*Protocol*$_1$, *Protocol*$_2$) :- *logical_theory*(*Protocol*$_1$, *Log*$_1$),
logical_theory(*Protocol*$_2$, *Log*$_2$), *same_theory*(*Log*$_1$, *Log*$_2$),
same_theory(*Log2*, *Log1*).
same_theory([ ], [ ]).
same_theory([*Head*$_1$| *Tail*$_1$], *Log*$_2$) :- *implies*(*Head*$_1$, *Log*$_2$), !,
same_theory(*Tail*$_1$, *Log*$_2$).

The logical theory of *Protocol*$_1$ is *Log*$_1$ and *Log*$_2$ is the logical theory of *Protocol*$_2$. The predicate *same_theory*(*Log*$_1$, *Log*$_2$) succeeds if each rule in the theory *Log*$_1$ can be derived from the theory *Log*$_2$. The base case of the predicate *same_theory* asserts that

empty theories are equivalent. The predicate *implies*($Head_1$, Log_2) returns *true* if the rule $Head_1$ can be derived from the logical theory Log_2 i.e. $Log_2 \vdash Head_1$.

6.18.5 Is a protocol a subset of another protocol

One protocol may be a subset of another protocol. Let agent *X* know protocol p_1 and agent *Y* know protocol p_2. If protocol p_1 is a subset of p_2, then both agents can interact according to p_1. Agent *Y* knows a richer protocol, p_2, but also knows all the states and processes that are possible in p_1. Agent *Y* knows protocols p_2 and p_1. The agents may adopt p_1 as the common protocol for their negotiation since all of them know p_1.

If a protocol, p_1, with logical theory Log_1, is implied by another protocol, p_2, with logical theory Log_2, then protocol p_1 is a subset of protocol p_2. $(Log2 \vdash Log1) \rightarrow (p_1$ is a subset of $p_2)$ holds. In a prolog-like notation, the predicate *sub_protocol* (p_1, p_2) returns *true* if protocol p_1 is a subset of protocol p_2.

sub_protocol (p_1, p_2) :- *logical_theory*(p_1, Log_1), *logical_theory*(p_2, Log_2),
sub_theory(Log_1, Log_2).

The predicate *sub_theory*(Log_1, Log_2) succeeds if all the rules in Log_1 can be derived from Log_2. Let protocol p_1 consist of the list of formulae [$Head_1$| $Tail_1$]. We check whether each rule in p_1 is implied by Log_2.

sub_theory([ ], Log_2) :- *writenl*(" 1^{st} *protocol is a sub-protocol of* 2^{nd} *one*").
sub_theory([$Head_1$| $Tail_1$], Log_2) :- *implies* ($Head_1$, Log_2), !, *sub_theory*($Tail_1$, Log_2).

6.19 Summary

This chapter defines axioms in ANML for various properties of a protocol and also shows how to prove them for a bilateral protocol in ANML. Other protocols and properties may be similarly represented and analysed. Additional properties include efficiency, performance or game theoretic properties such as Pareto efficiency, stability, possible equilibrium, deception-free and conflict resolution. See Rosenschein and Zlotkin, [1998] for more on game theoretic properties. The proof

methods in this chapter may also be used for analysing process executions in state-transition-like systems.

Properties can help to choose between protocols. However proving that a protocol satisfies a property may not always be feasible. This problem may be countered by monitoring the executions of a negotiation for violations of the property. While it is possible to monitor an interaction for violations of safety properties, this is not so easy for liveness properties. Equivalence between protocols are also discussed. We have defined a consistency property for a protocol, but inconsistency in a negotiation state may still arise because of discrepancies between the beliefs of a group of agents. This is the subject of the next chapter for reasoning about the knowledge and beliefs of a group of agents.

7 Reasoning about a Group's Beliefs

7.1 Introduction

Agents choose to negotiate because it is more cost-effective or it is the only way to achieve their goals. Negotiations may be collaborative or competitive. A negotiation may be competitive but still require the agents to collaborate according to a protocol. A participant does not necessarily know the beliefs and goals of other agents. There are various forms of beliefs and knowledge in a multi-agent system, for example common beliefs, common knowledge, private beliefs, private knowledge, joint beliefs and joint knowledge. Knowledge is taken as what is certain and a fact whereas belief is subjective to an agent and may be untrue. Cohen and Levesque [1991] describe theorems for how a society of agents can co-operate in a dynamic environment to achieve a goal through mutual beliefs, mutual goals, joint commitments and joint intentions. The group can monitor the success or failure of the joint effort.

This chapter discusses the mental states of negotiating agents. Potential divergence between mental states about a shared formula may lead to inconsistent worlds and necessitates communication. First this chapter analyses the effect of an agent extending a protocol on the common knowledge of the group. Secondly, the need for a synchronisation protocol is addressed to ensure that all participants attain mutual beliefs about the state of a negotiation after each message exchange, independent of the quality of the communication layer underneath.

7.2 The Theory of Joint Intentions

Cohen and Levesque, [1991], describe a formal theory for teamwork, called the theory of joint intentions. This section describes Cohen and Levesque's theory as it relates to

the mental states of the participants in a negotiation. Agents in a group may possess the right intentions and beliefs about each other, but absence of mutual beliefs about agreements leads to misunderstanding. In Cohen and Levesque's joint intentions theory, agents first form joint intentions to act in the future, keep those joint intentions over time and then jointly act. It becomes common knowledge that the agents are committed to their intentions, thereby discouraging any change in goals and desires on everyone's part.

Joint Intention

The definition of joint intention guarantees that goals and intentions persist over time and that the mutual knowledge about these goals and intentions will persist. Agents *X* and *Y* jointly intend to do some action iff they mutually know about their intentions, their joint intentions and this mutual knowledge persists until it is mutually known that the action is no longer needed.

Joint Commitment and De-commitment

A joint intention can be defined in terms of a joint commitment for doing a collective action, with the team acting in a joint mental state. An agent does not keep to itself private beliefs that a team's goal is no longer valid, rather it must make this fact mutually known to the other participants. When an agent discovers a joint goal to be impossible, its goal becomes to make it known to the others that their goal is impossible so that the team as a whole may give up their joint intention and goal. Individual intentions persist as long as joint intention does.

Jointly intending sequential actions

An agent may know it is executing a repetitive or sequential action in a shared plan, but it need not know its number in the sequence or if other steps in the sequence have started or are over. An agent can execute a sequence of actions knowingly without being aware of executing the individual steps. Each agent should however recognise its turn and what it is supposed to do. It should be possible to stipulate conditions for the execution of a sequence.

7.3 Knowledge and Belief Systems

Knowledge and belief have long been studied in epistemology, philosophy of mind and in philosophical logic. Belief frequently applies to a sentence one is unsure about or for which there is insufficient proof. On the other hand, knowledge applies to things that are true. Knowledge may be regarded as true belief. Knowledge and beliefs may be treated as modalities through epistemic logic and doxastic logic. The beliefs of an agent are formalised in terms of an accessibility relation over possible worlds. Beliefs are the propositions that are true in all the accessible worlds. Epistemic logic introduces modal operators for knowledge. The formula $K_i\,\alpha$ may be read as agent *i* knows that α and the formula $B_i\,\alpha$ may be read as agent *i* believes α. See [Meyer and Van der Hoek 1995] for a discussion on epistemic logic. The formula $E_G\alpha$ denotes *everyone in a group of agents, G, knows* α. Let the formula $Bel_G\alpha$ denotes *everyone in a group of agents, G, believes* α. The formula $C_G\alpha$ denotes α *is common knowledge among the agents in G*. $C_G\alpha$ can be expressed in terms of $E_G\alpha$, where $C_G\alpha$ is true if everyone in *G* knows α, everyone in *G* knows that everyone in *G* knows α, etc.

$$C_G\alpha \equiv E_G\alpha \wedge E^2{}_G\,\alpha \wedge \ldots \wedge E^m{}_G\,\alpha \wedge \ldots\ldots$$

where $E^1{}_G\alpha = E_G\alpha$ and is defined as everyone knows α. $E^{k+1}{}_G\,\alpha = E_G\,E^k{}_G\,\alpha$, *for* $k > 0$. Similarly, let the formula $CBel_G\alpha$ denote common or mutual beliefs where $CBel_G\alpha \equiv Bel_G\alpha \wedge Bel^2{}_G\,\alpha \wedge \ldots \wedge Bel^m{}_G\,\alpha \wedge \ldots\ldots$ $CBel_G\alpha$ is true if everyone in *G* believes α, everyone in *G* believes that everyone in *G* believes α, and so forth. $Bel^{k+1}{}_G\,\alpha = Bel_G\,Bel^k{}_G\,\alpha$, *for* $k > 0$. Common knowledge can be regarded as true mutual belief

An axiomatic system for knowledge and belief in terms of axioms *K*, *D*, *T*, *4* and *5* may be defined. See chapter 3 for definitions of these modal axioms. The *K-axiom*, $(K_i\,\alpha \wedge K_i\,(\alpha\rightarrow\beta)) \rightarrow K_i\,\beta$ for $i = 1..m$, states that knowledge of agent *i* is closed under classical logical consequence.

The axiom *D* states that an agent's beliefs are non-contradictory. The axiom can be written as $K_i\,\alpha \rightarrow \neg\, K_i\,\neg\alpha$ for knowledge and $B_i\,\alpha \rightarrow \neg\, B_i\,\neg\alpha$ for beliefs. If agent *i* knows or believes α then it does not know or believe the contrary, ¬α.

The axiom *T*, ($K_i \alpha \rightarrow \alpha$), is often called the *knowledge* axiom, since it distinguishes knowledge from belief and applies only to knowledge. Axiom *T* states that what is known is true. Knowledge is thus often defined as true belief: agent *i* knows α if *i* believes α and α is true, [Wooldridge and Jennings 1995a].

Axiom *4* is called the *positive introspection axiom*: $K_i \alpha \rightarrow K_i K_i \alpha$. An agent knows what it knows and similarly for beliefs. Introspection, [Konolige 1986], allows examining one's own beliefs.

Axiom *5* is the *negative introspection axiom*: $\neg K_i \alpha \rightarrow K_i \neg K_i \alpha$ and similarly for beliefs. It states that an agent is aware of what it doesn't know. Axiom *5* is controversial philosophically since if an agent ignorant of something, it is unlikely in general that it knows (or believes) this ignorance, [Meyer and Van der Hoek 1995].

Positive and negative introspection together imply an agent has perfect knowledge about what it does and doesn't know. It is generally accepted that positive introspection is a less demanding property than negative introspection, especially with limited resources for reasoning. The axioms *KTD45* are often chosen as a logic of *knowledge*, and *KD45* as a logic of *belief*. However an *S4* system may also be accepted for reasoning about knowledge and similarly axioms *KD4* for reasoning about beliefs.

7.4 Knowledge and Belief in a negotiation

Both knowledge and belief are used in our approach for reasoning about negotiation. The protocol is considered common knowledge in a group of agents who are encouraged to comply with it so as to reach a desired negotiation state. On the other hand, the negotiation state is taken to be part of the mutual beliefs of a group. Messages between agents may get lost or delayed such that an agent's beliefs about the negotiation state are not necessarily true and may not form part of other agent's beliefs. Another reason for using belief is that a group of agents may negotiate about a subject they all believe in but which is not in fact true. For example, two agents may believe they are negotiating about Christmas turkeys according to an English auction, when in fact they are negotiating about pigeons. It could be said that the

same applies to the interaction protocol where the agents believe they commonly know they are complying with an English auction, but in fact they are complying with a Dutch auction. Using this stance leads to an agent not knowing anything and believing that it knows some things. We draw a line and consider the interaction protocol as knowledge, and the subject and state of negotiation as beliefs. Beliefs are preferred to knowledge in our framework when representing and reasoning about the state of negotiation according to a group of agents. Even if a group's beliefs are not true and they are not aware of it, the negotiation is still valid to the agents, suits their purposes and achieves their goals.

7.4.1 Example of Contents of Knowledge and Belief in a negotiating group

In a negotiation, the common knowledge and beliefs of a group of agents may consist of a number of sentences. In addition, an agent may privately possess individual knowledge and beliefs. Below are listed some of the elements that may be common or private knowledge or belief between negotiating agents.

Common Knowledge

- The protocol followed by the group for a negotiation
- The possible roles the agents can take.
  Roles can embed constraints about permissions and capabilities. Roles are defined in a protocol.
- If applicable, the institution or market-place supporting the negotiation.
- Commitment
  There may be some states in a protocol that involves a higher degree of commitment i.e. they are nearer to a successful terminal state. For example, in a bilateral protocol *proposed* and *offered* reflects more commitment than *requested* state. The state *agreed* is a committed state.

Common Beliefs

- The subject of negotiation
  The subject of negotiation may consist of a list of attributes stored as name-value pairs e.g. [(*article*, *Jeans sale JB007*), (*price*, *£50*), *offered*(*john*)].
- The set of agents involved in the negotiation.

An agent does not know the number of participants in an open negotiation as in an auction.

- The state of the negotiation is mutually believed by the group.
- The history of the negotiation

  The history of the negotiation depends on the states reached and actions that have been performed. The history of the actions and state transitions may be kept for learning purposes, non-repudiation, recovery, etc.
- The group's goals
- The etiquette

  An etiquette could ensure that agents reply when required, have resources to back their commitments and respect any agreement or contract. It could also embody sincerity and fairness aspects in interacting. A agent may wrongly believe an etiquette.
- Contract and obligations

  If a negotiation has successfully terminated, there may be a contract between the participants. The contract may require its execution as in the promissory protocol in chapter 4. It may also involve liabilities in the case of non-execution. The contract may be derived from the state of a negotiation.
- Public conditions and variables

  There may be other conditions that may be commonly believed to a group of agents e.g. the time elapsed and remaining, beliefs about the environment or constraints that an agent choose to make public.

Individual Beliefs and Knowledge

This section gives an indication of the possible contents of an agent's individual beliefs in a negotiation. Some of the elements may be derived from the common beliefs and knowledge, for example planning and analysis of the history of the negotiation. An agent may need information from one negotiation to influence its decisions in another negotiation.

- The state of an agent

  The state of an agent reflects the state of affairs on its side. It may depend on its actions and the mutual beliefs.

- Goals and sub-goals

  It is up to an agent to choose and update its goals and sub-goals. Goal-oriented behaviour is specific to an agent as it chooses when and which goals to satisfy.
- Valuation functions, reservations, estimates (of costs, utility)

  An agent may assign values to costs and utilities for the attributes in the subject of a negotiation. A valuation function may help to do so.
- Strategies

  An agent can calculate its strategies through decision-making mechanisms, heuristics algorithms, risk analysis, statistical estimates and game theoretic, path-finding and planning techniques. It may have an ordered list of paths for planning.
- Beliefs about other agents
- Learning functions
- Trust

  This includes the trust of an agent towards other participants, the market-place or the institution. Trust can be built from several sources such as other negotiations, the history of that negotiation and third parties.
- History of various negotiations

  An agent has beliefs about past and parallel negotiations.
- Private constraints

  There may be constraints private to an agent e.g. about time and resources or its environment.
- Predictions

  An agent may form predictions and construct plans using its estimates, strategies and its beliefs.

7.5 The Properties of a Theory

The previous chapter specifies various safety and liveness properties of a protocol expressed as a theory. Formal properties such as satisfiability and consistency of a theory can also be derived. The theory of a protocol is common knowledge in a group of negotiating agents. Therefore analysing the properties of the theory of a protocol helps in reasoning about the common knowledge of the group regarding that protocol. The common knowledge in a group G about the protocol is referred to as the *common*

theory in group *G*. Consistency of the common theory regarding the protocol entails consistency between the agents' knowledge about the protocol. Similarly the properties of the common theory reflect on the properties of the common knowledge about a protocol. This section describes the properties of a theory using the bilateral protocol as example.

Language

A language is an enumerable set of non-logical symbols, i.e. names, function symbols, sentence letters and predicate letters. A sentence or a formula in a language is one whose non-logical symbols all belong to the language. ANML is a language.

Theory

A theory is a set of sentences in some language that contains all of its logical consequences that are sentences in that language. Protocols are represented as theories in ANML.

A Complete theory

A theory *T* is complete if for every sentence *A* (in the language of *T*), either *A* or ¬*A* is a theorem of *T*. The theory of a protocol is complete if there are no undefined states.

To prove that the theory of the bilateral protocol is complete, the truth-values of all its possible states are checked before and after each state transition. The possible states in a bilateral negotiation are *negotiating, requested*, *offered, proposed, open, closed, agreed, rejected*, *timedout* and their negation. The bilateral protocol is complete if all its states remain well-defined (either they are true or false) before, during and after a negotiation, therefore at all times. The proof is similar to the structural induction proofs in the last chapter.

A consistent theory

A consistent theory contains no theorem whose negation is also a theorem.

A satisfiable theory

A theory is satisfiable if it has a model. A consistent theory is satisfiable.

Extension and conservative extension of a theory

A theory *T* is called an extension of theory *S* if *S* is a subset of *T*, i.e. if any theorem of *S* is a theorem of *T*. Theory *T* is a conservative extension of theory *S* if every sentence of the language of *S* that is a theorem of *T* is a theorem of *S*.

As an analogy, consider an extension *T* of the bilateral protocol *S*. The following two theorems are part of the theory *S*:

$$\neg open \leftrightarrow \text{none\_of}(\{ requested(X), offered(X) \})$$
$$open \leftrightarrow \text{one\_of}(\{ requested(X), offered(X) \})$$

The theory *T* only contains the theorem (none_of ({ *requested*(*X*), *offered*(*X*) }) ∨ one_of ({*requested*(*X*), *offered*(*X*) })) in addition to the theorems in *S*. The additional theorem in theory *T* is implied by the two above theorems in theory *S*. *T* is thus a conservative extension of theory *S*.

7.6 Common, Individual and Joint Theory

Consistency in the individual and common knowledge and beliefs is ensured by the *D-Axiom* if an *S4* or *S5* system is used.

7.6.1 Common Theory

A theory α is defined to be *common theory* in a group of agents *G*, if α is common knowledge in the group *G*. The theory of a protocol in ANML is a common theory in group *G* if it is commonly known by the agents in *G*. All the agents in the group know the protocol and know that the other agents know it and so on. If Cohen and Levesque's joint intentions theory applies in *G*, then an agent *X* intends to follow the common theory, knows that others in the group follow the protocol and that others know that *X* follows the protocol.

This thesis assumes the theory of a protocol to be common knowledge in a group during a negotiation. A protocol can become common knowledge on joining the

group, by learning from a repository of protocols, from a pre-negotiation phase or from being advertised by the institution supporting the interaction e.g. a market place.

7.6.2 Individual Theory

Consider two agents who know two different languages (L_1 and L_2) and two different theories, T_1 and T_2, where L_i is the language of theory T_i. They need a common language and a common theory to negotiate. Let T_0 be the *common theory* where T_0 is the set of sentences in a common language $L_0 = (L_1 \cap L_2)$. Each agent's *individual theory*, T_1 or T_2, is an extension of the common theory T_0. If T_0 is not empty then both agents have some common grounds, T_0, on which to interact.

The *individual knowledge* of an agent in a group is private to that agent. *Individual knowledge* and *individual theory* at an agent denotes the same concept. Various negotiations and protocols, transactions, strategies and accumulated experience may form part of an agent's individual knowledge and beliefs. In addition to knowing the common theory (i.e. protocol), an agent can privately extend the common theory in order to guide its responses. For example in a negotiation following the bilateral protocol, an agent called *Sam*, adds the following rule to its individual theory: *proposed*(X) $\rightarrow$ [*Sam.request*] *requested*(*Sam*). Sam decides to respond with a *request* after any proposal. This theorem is not part of the common theory of the group, but is part of Sam's individual theory.

In a negotiation, all agents in a group know the common theory and have possibly different individual theories. The properties of these theories can be analysed. The responsibility for designing and verifying the properties of a common theory or protocol would probably lie with the provider of the protocol or the institution supporting the negotiation. On the other hand, an agent manages its individual theory and beliefs about a negotiation. Figure 7.1 illustrates possible categories in an agent's individual knowledge. It includes the common theory about negotiation X, and its private knowledge about X and about its other transactions. The agent would have the same elements for its individual beliefs.

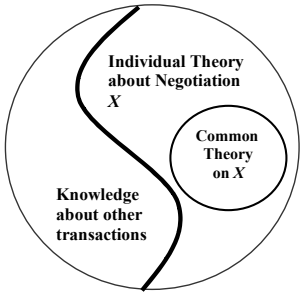


Figure 7.1 An agent's individual knowledge

Agent 1
Language L_1, Theory T_1

Agent 2
Language L_2, Theory T_2

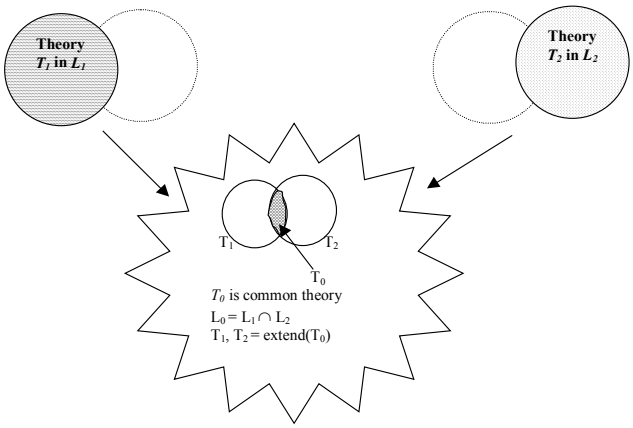


Figure 7.2 Common theory T_0 and individual theories T_1 and T_2

7.6.3 Joint Theory

The joint theory of a group of agents follows from the consequence of the union of their individual theories. Usually, no agent knows the joint theory except if it has a view of all of the agents' individual theories or if the joint theory is equivalent to the common theory. Consider two agents who know different languages (L_1 and L_2) and different theories, T_1 and T_2, where L_i is the language of theory T_i. As discussed above, $L_0 = (L_1 \cap L_2)$ and T_0 are the common language and theory respectively and an agent's individual theory is an extension of the common theory. The *joint theory*, T_3, is defined to be the set of sentences, in language $L_3 = (L_1 \cup L_2)$, that are consequences of $(T_1 \cup T_2)$. T_3 is an extension of the common theory T_0 and of the individual theories, T_1 and T_2. If T_1 and T_2 are conservative extensions of T_0, then T_3 is also a conservative extension of T_0.

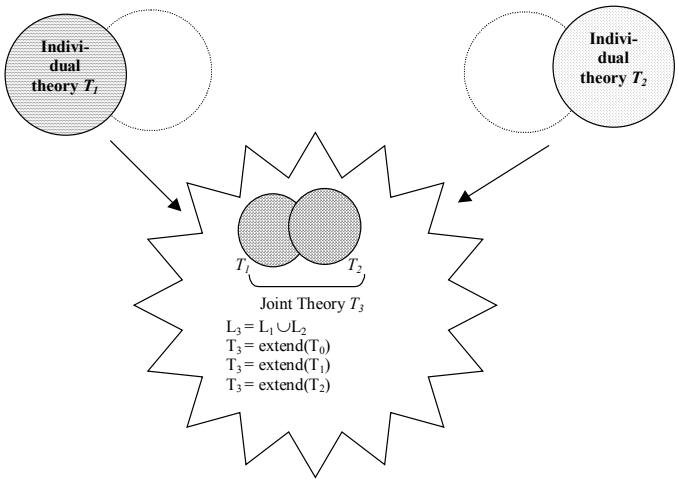


Figure 7.3 Joint Theory

7.7 Consistency of the Joint Theory

The protocol to be followed in a negotiation is known to all the participants. In addition, the agents may have their own theory about how best to negotiate. Inconsistency may arise in the joint theory of a group even though the common and individual theories are consistent on their own. For example, in a bilateral negotiation, one agent may believe the current state to be *agreed* while another agent believes it to be *timedout*. The states *agreed* and ¬*agreed*, *timedout* and ¬*timedout* are all *true* in the joint theory. Their individual theories are thus consistent on their own but inconsistent with each other. This leads to an inconsistent world where for one agent the negotiation is a success while the other thinks it is a failure.

The possible reasons for inconsistency in the joint theory regarding a protocol include the common theory being incorrect or the additional theorems in an agent's individual theory render it inconsistent with other agents' individual theory. A group of agents may follow a common protocol but individually extend the protocol for their purposes. Inconsistency may arise between the different individual extensions of the protocol. The issue is, given a common theory and individual theories in a group how is consistency ensured in the joint theory regarding a particular negotiation. Robinson's consistency theorem, [Boolos and Jeffrey 1989], is applied to ensure consistency of the joint theory. Let Agent *i* have individual theory T_i. The reasoning and conditions for maintaining consistency are given below.

Robinson's consistency theorem

If L_i is the language of T_i (i= 0,1,2), $L_0 = (L_1 \cap L_2)$, T_0 is a complete theory, and T_1 and T_2 are satisfiable extensions of T_0 (which is therefore satisfiable), then $T_1 \cup T_2$ is satisfiable.

The joint theory of T_1 and T_2 is derived from $(T_1 \cup T_2)$. L_0 is the common language of L_1 and L_2. T_0 is the common theory of T_1 and T_2. If agent i has individual theory T_i, then the joint theory of a group of two agents is satisfiable if Robinson's consistency theorem applies. This rule can be extended to apply to a group of n agents with individual theories.

Extension of Robinson's rule: If L_i is the language of T_i (i= 0,1,2,…,n), $L_0 = (L_1 \cap L_2 \cap \ldots \cap L_n)$, T_0 is a complete theory, and T_1, T_2 up to T_n are satisfiable extensions of T_0 (which is therefore satisfiable), then the theory $(T_1 \cup T_2 \cup \ldots \cup T_n)$ is satisfiable.

Proof by induction

The base case is covered by Robinson's consistency theorem for two agents in a group with individual theories T_1 and T_2.

Induction Hypothesis

Assume that the extension of Robinson's rule holds for theories T_1 to T_k i.e. k agents in a group, where agent i has individual theory T_i.

Induction Proof

The joint theory of $k+1$ agents is proved to be satisfiable from the preconditions of Robinson's rule for $k+1$ agents. Let L_i be the language of T_i (i= 0, 1, 2,…, $k+1$, .. m).

Preconditions:

- The common theory, T_0, of T_1, T_2 up to T_{k+1} is in the common language $L_0 = (L_1 \cap L_2 \cap \ldots \cap L_{k+1})$.
- Theories T_1, T_2 …T_{k+1} are satisfiable extensions of T_0.
- T_0 is complete and satisfiable.

The joint theory of T_1, T_2 up to T_{k+1} can be derived from ($T_1 \cup T_2 \cup \ldots \cup T_{k+1}$). From the induction hypothesis the joint theory of T_1, T_2 up to T_k is satisfiable. That is, theory ($T_1 \cup T_2 \cup \ldots \cup T_k$) is satisfiable. Since T_1, T_2 $\ldots T_{k+1}$ are satisfiable extensions of T_0, then the theory ($T_1 \cup T_2 \cup \ldots \cup T_k \cup T_{k+1}$) is an extension of T_0. T_0 is complete and T_{k+1} is a satisfiable extension of T_0. Then the theory ($T_1 \cup T_2 \cup \ldots \cup T_k \cup T_{k+1}$) is satisfiable. QED.

The conditions for a satisfiable joint theory are therefore:

1. T_0, the common theory, is a complete theory.
2. The individual theories T_1 … T_k are satisfiable extensions of the common theory T_0.
3. T_0 is a satisfiable theory. This follows from the above two conditions.

Hence the above three conditions must be satisfied for maintaining consistency in the joint theory between a group of agents.

7.7.1 How Robinson's rule preserves consistency of the joint theory

If L_i is the language of T_i (i= 0,1,2), $L_0 = (L_1 \cap L_2)$, T_3 is the set of sentences of ($L_1 \cup L_2$) that are consequences of ($T_1 \cup T_2$), and T_1 and T_2 are conservative extensions of T_0, then T_3 is also a conservative extension of T_0, [Boolos and Jeffrey 1989].

According to the above theorem, if the individual theories of a group of n agents (T_1 … T_n) are conservative extensions of the common theory (T_0), then the joint theory (T_k) is also a conservative extension of the common theory (T_0), where T_k is the set of sentences from ($L_1 \cup \ldots \cup L_k$) that are consequences of ($T_1 \cup \ldots \cup T_k$).

A satisfiable extension of a complete theory is conservative and a conservative extension of a satisfiable theory is satisfiable. The common theory is satisfiable and the joint theory is a conservative extension of the common theory and is thus satisfiable. By the same token, the common theory is complete and satisfiable, and the joint theory is a satisfiable extension of the common theory and is conservative.

7.7.2 Maintaining Consistency in Negotiation

Interpreting Robinson's consistency theorem in the context of a negotiation yields: If the theory of a protocol is complete and the agents' individual theories about the protocol are satisfiable extensions of the protocol, then the joint theory about that protocol is also consistent.

Therefore for a consistent joint theory in a negotiation, the protocol must be complete. If a theory is complete, it implies any satisfiable extension. An agent's individual theory must be a satisfiable (and hence conservative) extension of the protocol. Any consistent extension that an agent privately wishes to add to a protocol should be derivable from the protocol. The developer of a protocol or the institution enforcing a protocol may be responsible for building and verifying that the theory of a protocol is satisfiable and complete. On the other hand, each agent must ensure that any extension it privately adds to the protocol is satisfiable and thus conservative. There is a shared responsibility for the joint theory to remain consistent.

The bilateral protocol has been proved to be consistent, complete and satisfiable. Therefore it meets the requirement of Robinson's consistency rule for a complete and satisfiable common theory. The onus now lies on the two agents' side to ensure that their individual theories are satisfiable extensions of the bilateral protocol. Consider as an example an instance of negotiation between two agents following a protocol. The different steps for preserving consistency in the joint theory are enumerated:

1. Define the common theory in the negotiation.
   The protocol is common knowledge and thus common theory between the two agents.
2. Verify that the common theory is complete and satisfiable.
   The theory of the protocol must be checked for being complete and satisfiable. The properties of the protocols can also be analysed.
3. Define each agent's individual theory (T_1 and T_2) as extensions of the protocol.
   The common theory can be derived from each agent's individual theory. An agent's individual theory of the protocol extends the protocol. For example, in a bilateral negotiation, an agent may privately decide to accept any proposal.

4. Verify that T_1 and T_2 are satisfiable throughout the negotiation.
   Each agent's individual theory about the protocol must be consistent and conservative extensions of it.
5. Apply Robinson's rule to show consistency of the joint theory.
   If all the above conditions are met, then according to Robinson's rule, the joint theory between the two agents regarding the protocol is consistent.

7.8 Beliefs about the Negotiation State in a Fallible Communication Medium

Agent interaction in realistic applications is subject to many forms of uncertainty - *information and network uncertainty, trust of and conflicts with other participants, lack of stability in a deal and risks about agreements and commitments, etc*. However, one of the most common forms of uncertainty occurs when a group has divergent beliefs about the interaction - some agents believe an agreement has been reached, while others believe it has been rejected or that they are still bargaining. Such misunderstandings can arise because of loss of network performance, spurious connections, message loss or delays.

We develop synchronisation protocols for a group of agents to attain the same beliefs about an interaction, *independent of the reliability of the underlying communication layer*. This section includes and proves theorems about a group's mutual beliefs, on which the safety of an interaction relies. Specifically, protocols for message exchange and belief revision and our reasoning for reachability of states during interactions are presented. Each protocol is proved to show that an increasing level of mutual and consistent belief is reached, thereby guaranteeing an interaction's integrity.

7.8.1 Defining the Problem

Issues arise when unexpected events occur, for example agents do not comply with or misunderstand the interaction protocol, or the communication is faulty. To deal with such issues and the semantics of agent communications, we regard an interaction as a joint process between agents. The steps in such a joint process progress by virtue of the propositions believed by the group. We use *belief*, instead of *knowledge*, because the property of *knowledge* being true means it is far harder to attain in practical

contexts than beliefs. Beliefs only require consistency and we assume this in our solutions. For example, an agent *X* may believe an agreement has been reached with agent *Y* when in fact it has misunderstood or mis-implemented the protocol, or there has been a security breach with a malicious agent impersonating *Y*, or *X* and *Y* do not share the same ontology. When these uncertainties are compounded with network unreliability, it is easier for an agent to believe some state than to know that state.

While an interaction protocol may be considered as common belief in a group of agents, the participants' individual beliefs about the progress of a particular interaction are liable to differ between the time a message is sent and received. For example, a sender has added beliefs about the message it sent compared to another agent which has not received that message. Therefore, some degree of common beliefs and consistency in joint beliefs about the interaction state are necessary to safely progress in an interaction and to avoid contradictions and any confusion in the group that may arise from a participant persisting with differing beliefs about the interaction state. This issue of consistency is an important problem because unreliable communication is the norm in the communication networks and infrastructures in which agents are most likely to be deployed. The key issue here is that the interaction must not continue without ensuring that all agents believe the same state at some point. Otherwise if messages are lost in an interaction according to protocol *P*, then one agent may believe the state to be *agreed* while another believes it to be *browsed*. Such discrepancies may lead to disputes worsened when monetary, time or safety-critical information are involved.

7.8.2 A Synchronisation Layer

Given these requirements, we develop *synchronisation protocols* that should be followed by all agents for message exchange and belief revision in an interaction. These protocols lie in a layer below interaction protocols but above communication (network) protocols (see Figure 7.4). Such synchronisation protocols specify the steps for sending messages, acknowledgments and for belief update. Their aim is to produce a degree of shared belief, if not common belief, about the state of an interaction, that is sufficient to remove uncertainty about that state. Moreover, we prove that our synchronisation protocols allow a group to attain the same beliefs about an interaction state before each transition and that an interaction safely terminates

with no uncertainty about its result. If the interaction has failed, then all agents believe so.

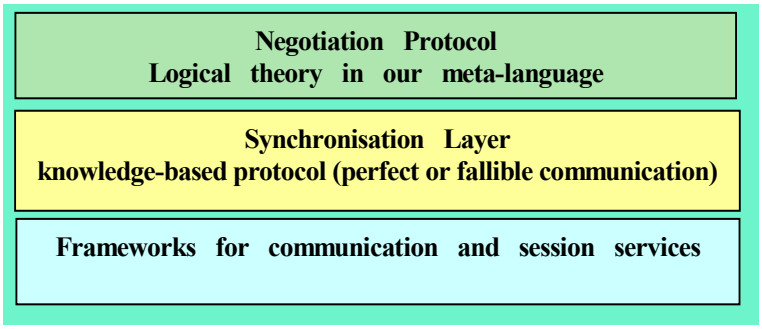


Figure 7.4 Various Protocols

Three types of protocol are distinguished in our approach – low-level, synchronisation and interaction protocols, as in Figure 7.4

Examples of low-level protocols are TCP/IP, HTTP or those defined by CORBA, JAVA RMI, SSL, for connection and session purposes. Interaction protocols constitute one of the main aspects of this thesis. Interaction protocols are used in high-level agent conversations with a plethora of underlying frameworks providing location, platform, session, communication and programming language transparency. Interaction protocols, such as the bilateral protocol, indicate possible sequences of ACL-like (but not only) messages e.g. a *request* may be followed by an *offer*. An interaction protocol is considered as common knowledge in a group complying with that protocol. The current state of a negotiation varies depending on the actions of an agent following the interaction protocol, for example from *browsed* to *offered* and then to *agreed*.

As a negotiation progresses, all the participants must individually believe and believe that their opponents believe the same state of that negotiation to avoid contradiction and confusion. While in a perfect communication medium, this mutual belief follows from the exchange of messages, this is not guaranteed when communication is faulty, involving lost, unordered messages and delays. We adopt an epistemic approach to address the problem of ensuring that there is a degree of shared belief, if not common belief, between the participants about the negotiation state when communication is

imperfect. The agents in a group follow a synchronisation protocol guiding how messages and acknowledgements are sent and how an agent revises its beliefs about the state of a negotiation. The synchronisation protocols in sections 7.12, 7.14 and 7.15 aims to remove inconsistency in the joint beliefs of a group of agents because of communication problems and aims to attain a degree of mutual belief. We wish to prevent one agent from believing a negotiation to be in state *x* while another agent believes it to be in state *y* and the negotiation continues without ensuring that both agents will eventually believe the same state at some point.

It has been argued that common knowledge is hard to achieve when communication is faulty, [Meyer and Van der Hoek 1995]. This chapter discusses this problem in the context of an automated negotiation. Since belief imposes weaker constraints than knowledge, a degree of common belief about the state of a negotiation may be attainable. The proposed solution incorporates a synchronisation layer between an interaction and a communication framework. Using a synchronisation protocol, we consider reaching a degree of shared beliefs about the current state, while beliefs about other participants gradually augments as negotiation progresses. Interaction and synchronisation protocols are assumed to be common knowledge within a group of agents. Section 7.11 describes the assumptions, conventions and conditions for message exchange and state transition during a negotiation and for its termination. The synchronisation protocol removes uncertainty about the exchange of messages at the end of intermediate steps in an interaction. Section 7.13 proves by induction that after any state-changing message, all participants following the synchronisation protocol have the same belief about the current state of that negotiation before the next transition. Finally this approach is illustrated through a generic scenario between two agents to show the evolution of their beliefs, predictions, message exchange and state transitions in a negotiation. The interaction process between these agents concludes with both of them believing the same terminal state and believing that their opponents believe so too.

7.8.3 The State of a Negotiation

As discussed in chapter 3, the state of an agent is different from what an agent believes about the state of a negotiation. The state of an agent remains private to that agent. Each agent has its individual and private beliefs useful for strategic reasoning

in competitive scenarios. Shared protocols are assumed to be common knowledge, non-deterministic and can be used to achieve a goal state.

The state of a negotiation changes depending on the actions of the agents in a group. The group adheres to a common protocol and should be aware of the state of a negotiation and of possible successor actions and states, given by the protocol. A current state means that state which is true while other non-parent states are false. A state of a negotiation is part of an agent's beliefs. Changes in beliefs about the state of a negotiation occur in the individual beliefs of an agent on sending or receiving messages. These changes are eventually propagated to the shared beliefs between the group of agents as the interaction continues. The term *shared belief* is used to indicate a degree of common belief, as explained in section 7.10. In order for each agent to achieve the same belief about the negotiation state and belief that its opponents believe the same state and so forth require coordination and allowances for faulty communication. We specify protocols, in the context of automated negotiation, for synchronising message exchange and update of private beliefs when it is possible for messages to be lost or arrive unordered.

7.9 Contribution to Communication and Interaction

This work on synchronisation advances the state of the art in multi-agent interactions; by developing an approach that ensures their safe progression through a number of identifiable states and termination in a consistent manner.

7.9.1 Practical Usability

Existing multi-agent system work in this area typically relegates this problem to the ISO transport layer and assumes that messages safely reach their destination. However, this is inappropriate in situations in which an agent has to choose compensating or alternative actions in cases of network lag or failure. For example, let an agent X send an agreement to n agents and then receive a confirmation from m agents ($m \leq n$). If any one of the n agent's confirmations are lost through the network, the group as a whole has to coordinate to ratify X's agreement and inform the others of the ratification, let alone deal with delayed receipt of messages. Thus, because perfect communication cannot be guaranteed in most multi-agent systems, the problem of

reasoning about message exchange is not strictly a transport layer issue. On the other hand, the issue of sending messages and acknowledgements should not be dealt at the interaction layer because at this level an agent does not want to concern itself with the intricacies behind message exchange. The interaction level should rather concentrate on higher-level reasoning. Consequently, we develop the concept of synchronisation protocols to lie between interaction and communication protocols. Moreover, we reason about beliefs rather than knowledge because common beliefs are easier to attain than common knowledge and are more appropriate. Common knowledge can be impossible to reach when communication is imperfect.

Our synchronisation protocols assume the completeness of the (common) interaction protocol. Such completeness may be enforced by the protocol's developer or the institution holding the negotiation. The interaction protocol may be pre-determined (which is possible in negotiation between companies, customers and providers, where privacy of information are important, our bilateral, multi-lateral or promissory protocols) and its completeness proved.

7.9.2 Related Work

Distributed database solutions to preserving the consistency of a shared resource are based on two-phase locking and commit protocols, [Özsu and Valduriez 1999]. They are not suitable for addressing the integrity of a group's knowledge here because the participants are autonomous agents instead of objects and servers. In distributed databases, timeouts ensure the integrity of data and rollbacks are used to restore the shared resource to its latest stored state. While it is feasible in a non-intelligent system to return to a previous state, this does not apply to a multi-agent system. The agents can collect information about a timed out or failed negotiation and their opponents. They can use this new information in case of timeouts to influence their behaviour in a restarted negotiation. Believing that a message did not get through allows the sender to rethink its strategy and send a different message from before. Therefore timeouts may give an advantage to a participant. Other events may also have occurred during the elapsed time before restarting a negotiation. In essence, the state of a multi-agent system on starting a negotiation is not equivalent to its state on restarting that negotiation if timed out.

Fagin and al., [1995], use epistemic logic in analysing protocols by reasoning about an agent's actions, its local states and its knowledge. More work on knowledge-based approach to protocols can be found in [Halpern and Zuck 1992, Neiger and Tutlle 1993].

The bit transmission problem, [Halpern and Zuck 1992], and end-to-end communication problem, discussed in [Fich 1998], are examples of communication problems in faulty medium, requiring synchronisation protocols. The bit transmission problem involves the uncertainty of a receiver *R* receiving a message (a bit) from a sender *S*, in a faulty communication line. If the channel is working then it is eventually known that the message is successfully received. Otherwise, if the communication channel is constantly faulty, one knows that a message will never be received. When it is not guaranteed when the channel is working, then messages may be lost in either direction at uncertain times. A medium may even be nearly perfect with a small error rate. In either case of perfect or imperfect communication, questions remain about the attainment of common knowledge. See [Fagin and al. 1995 and Meyer and Van der Hoek 1995] for a discussion about common knowledge in transmitting messages over a network.

Fagin and al. [1995] describes a protocol overcoming the bit transmission problem. It dictates that *S* keeps sending its message until it receives an acknowledgement from *R*, while the latter keeps sending its acknowledgement after it receives the bit from *S*. *S* keeps sending its message since it does not know that *R* has received the bit. *S* knows so when it receives *R*'s acknowledgement. However *R* does not know if *S* has received its acknowledgement. Even if *R* stops receiving the bit from *S*, this may mean that the messages are lost rather than *S* has received *R*'s acknowledgement. Therefore *S* could send an acknowledgement in response to *R*'s acknowledgement, but this may escalate to *R* and *S* sending acknowledgements to each other. Faulty communication engenders this type of uncertainty in message passing systems. Fagin and al. [1995] analyses the bit transmission problem using the formalism of local states, actions and global states at the agents and channel sides and ascribing knowledge to the sender and receiver. Given a synchronisation protocol, the number of acknowledgements and counter-acknowledgements determines the point where an agent can eliminate uncertainty although the channel may be faulty. In the bit-

transmission protocol where a bit is being conveyed, three acknowledgements from *R* and *S* combined are sufficient to counter the uncertainty from a faulty channel.

End-to-end communication, also known as the sequence transmission problem, is the problem of sending a sequence of messages from one processor, the sender, to another processor, the receiver, through an unreliable communication network. This is an extension to the bit transmission problem. A sender *S* has an infinite sequence of data that it wants to transmit to a receiver *R* when communication is faulty. A protocol for exchanging these bits must exhibit safety and liveness properties for correctness. Solutions to the sequence transmission problem include the well-known ABP (Alternating Bit Protocol), [Bartlet and al 1969], Stenning's protocol, [Stenning 1976] and work by Aho and al., [1979, 1982]. The ABP is a connection-less protocol for transferring messages in one direction between a pair of entities in a situation where messages may be lost or duplicated, but never corrupted. Tagging messages with 0 or 1, and resending until an appropriately tagged acknowledgement is received may provide assurance of correctness. However, this protocol does not detect the case when two consecutive bits are lost or when bit (*i*+2) arrives before bit *i*. Stenning's protocol is designed for asynchronous systems where messages can be deleted, duplicated, reordered or detectably corrupted. Safety and liveness properties have been formulated and proved for Stenning's protocol in Hailpern and Owicki [1980]. [Fagin and al. 1995] give a knowledge-based analysis and solution from reasoning about the data communication and local states of the agents and the environment. *S* repeatedly sends the i^{th} bit to *R* until *S* knows that *R* has received it and its order in the sequence. *S* then sends the $(i+1)^{th}$ bit to *R*. Once *R* receives the i^{th} data, it requests *S* to send the next data. The semantics of first-order modal logic forces free variables to act as rigid designators. This is how Fagin and al. [1995] counters the problem of *R* not knowing that it is the i^{th} bit being sent. *R*'s request for the next bit is interpreted an acknowledgement for receiving the previous bit.

However the same principle cannot be used in a negotiation since a sequence of bits is not being sent but rather actions for triggering the state of a negotiation. These actions depend on the interaction protocol, are not ordered as a linear sequence and any agent may be a sender or receiver. Van der Hoek and Wooldridge's, [2002], solution to the sequence transmission problem requires *R* and *S* sending three

acknowledgements between themselves after each bit. After receiving the third acknowledgement, *R* sends the next bit. However this introduces the redundancy of two additional acknowledgements being sent after *S* receives the first acknowledgement from *R*. Sending the $(i+1)^{th}$ bit is an acknowledgement (second acknowledgement) in itself from *S*. Then acknowledging the $(i+1)^{th}$ bit is an acknowledgement (third) of the i^{th} bit from *R*. This idea of having implicit acknowledgements to previous messages, embodied in successor messages is used in our synchronisation protocol for reaching a state of shared belief that eliminates uncertainty.

7.10 An Interacting Group's Beliefs

For the sake of conciseness and familiarity, this paper henceforth uses for group beliefs the operator symbols *E* and *C* normally denoting group and common knowledge. This is allowable because we are redefining the semantics of the operators to denote beliefs utilizing preconceptions about knowledge operators. An axiomatic system for belief may be defined in terms of axioms for consistency and introspection (section 7.3). We assume that each agent in a group has such a system of beliefs and is aware that others do.

The formula $B_i\alpha$ is read as agent *i* believes α, $E_G\alpha$ is read as *everyone in a group of agents, G, believes* α and $C_G\alpha$ means α *is common belief among the agents in G* (where as for knowledge $C_G\alpha$ can be expressed in terms of $E_G\alpha$). With this standard definition, certain properties of protocols can be proved.

We assume the protocol of an interaction is part of the common beliefs of a group where all agents are aware of permissible states and actions and believe that other participants have the same beliefs. Thus an agent *X* may believe that it follows a protocol and believes that others in the group both believe and use the same protocol and believe what *X* believes about the protocol. The individual beliefs, B_i, of an agent *i* extend the common beliefs of the group and may include agent *i*'s beliefs about its interactions, environment and strategies. An agent revises its individual beliefs as the interaction progresses. Thus the individual beliefs of the agents in a group about the

history and state of an interaction may differ between each other while a message is being relayed.

7.10.1 Consistency of Joint Beliefs

The *joint beliefs* of a group are the sentences that are consequences of the union of the individual beliefs of the agents. Thus the joint beliefs are extensions of the individual and common beliefs of a group. The joint beliefs relate to the beliefs of all the agents in a group and unless all agents publicise their beliefs, no agent is privy to the joint beliefs of the group. Given the individual beliefs of the agents differ, the joint beliefs of a group need not be completely consistent.

However consistency of the joint beliefs *about an interaction state* is needed before progressing to the next state. Consider a multi-lateral interaction between n agents, where the interaction state is *motioned* (after raising a motion). This entails that all n agents believe the state to be (*motioned* $\wedge$ $\neg$*seconded*). An agent i sends a message m to second the motion to eventually trigger a *seconded* state. On sending m, agent i may believe the new interaction state is (*seconded* $\wedge$*motioned*). Agent i's belief about the interaction state is inconsistent with the other participants. On p agents receiving m, they update their beliefs about the interaction state to (*seconded* $\wedge$*motioned*). Now (p+*1*) agents beliefs are inconsistent with (n-p-1) agents. This is as expected given delays in message transfer over networks. It is important though that those (p+1) agents do not continue with the interaction with voting and agreement while some of the group still believe (*motioned* $\wedge$ $\neg$*seconded*), since this would result in agents receiving unexpected messages. An agent still believing the state to be $\neg$*seconded* could then receive a message about the result of all votes, which should only follow after a *seconded* and *voting* state.

Thus we propose synchronisation protocols to ensure that all agents believe the interaction state α at world w before proceeding on to the next state at world w_1, resulting in the formula $E_G\alpha$ holding in group G at world w. In fact an interaction state α emerges from it being a consistent and joint belief in a group at w. With the definitions of completeness and consistency, a protocol is complete if all states are well-defined, that is either a state or its negation can be derived at any instance. A

protocol is consistent if it does not allow a rational agent to believe both a state and its negation at any instance.

Theorem1 A complete and consistent protocol allows a group of agents to attain consistency in their joint beliefs.

Proof The proof of theorem1 is a ramification of Robinson`s Consistency theorem, see section 7.7.

Convention1 The formula $E_G\alpha$ holds about interaction state α in group G before any further transition to a subsequent state β.

In addition to consistency in joint beliefs, safe termination of an interaction requires that the final state becomes a common belief of all agents. For example, if an interaction terminates in an *agreed* state, each agent has to believe that everyone believes the same state and that everyone believes that it believes the state is *agreed*, etc. There is then no uncertainty about joint commitments if a commitment state like *agreed* is common belief. In such situations, an agent can fulfil its part of the bargain, believing that the other agents have the same beliefs as itself about the state. Ideally not only the terminal state but also the different states of an interaction should be common belief in a group. However in imperfect communication environments, common beliefs, as conventionally defined above, about each state before carrying on to the next state would require an indefinite number of acknowledgements and is unfeasible. Instead we propose to settle for shared beliefs which brings about common beliefs as argued in the following section.

7.10.2 Shared Beliefs

While not achieving conventional common belief about the state of an interaction in a group, we endeavour to ensure an increasing level of mutual belief to remove uncertainty about an interaction's progress. Following convention1, during an interaction, as long as the formula $E_G\alpha$ about interaction state α holds, the agents can continue on to the next state transition. Subsequent messages about successive state transitions implicitly increase the mutual beliefs about the state α being part of the

history of the interaction. As an example, let an interaction perform the succession of states 1 to 4 corresponding to (*requested ; proposed* ; *offered* ; *agreed*). At step 2, before a *proposed* state, the formula E_G*requested* holds. At step 3, before an *offered* state, the formula (E_G *proposed* $\wedge$ $E^2{}_G\delta$) holds where δ entails that *requested* was valid before the current state *proposed*. At step 4, before *agreed*, the formula (E_G *offered* $\wedge$ $E^2{}_G\,\gamma \wedge E^3{}_G\delta$) holds where γ entails that *proposed* was valid before the current state *offered*. After step 4, the state of interaction is *agreed* yielding the formula (E_G *agreed* $\wedge$ $E^2{}_G\,\lambda \wedge E^3{}_G\,\gamma \wedge E^4{}_G\delta$) where λ entails *offered* was a preceding state. In this way, the history of an interaction can be derived from the interaction protocol and the interaction state.

Definition: The state α is shared belief if everyone believes the state α and believes that everyone believes so too - ($E_G\,\alpha \wedge E^2{}_G$)

If *agreed* is a terminal state, then a group of agents must ensure that not only E_G *agreed* but also there is a degree of common belief about *agreed*. We settle for a group believing the first two terms of common beliefs which we call *shared belief* because shared belief about an interaction state allows the group to reach common belief, as shown in theorem2 below.

7.10.3 Sufficiency of Shared Beliefs

Theorem2 gives the conditions for deriving common beliefs about the state of an interaction in a group from it being a shared belief. If each agent is aware of the other agents' reasoning system, then through positive and negative introspection about the possible interaction states, the group can reach common beliefs about these states.

Theorem2 If the system of beliefs of all agents is consistent and is common belief in group *G*, if the interaction and synchronisation protocols are common in *G* and believed to be so, if group *G* attains shared belief about an interaction state α, then the interaction state α is common belief in group *G*.

Proof It is sufficient to prove ($E_G\,\alpha \wedge E^2{}_G\,\alpha \rightarrow E^3{}_G\alpha$) which results in ($E_G\,\alpha \wedge E^2{}_G$) $\rightarrow$ ($E_G\,\alpha \wedge E^2{}_G \wedge \ldots \wedge E^{m-1}{}_G\,\alpha \wedge E^m{}_G \wedge \ldots$). Assume that everyone believes that everyone believes the state is α i.e. ($E_G\,\alpha \wedge E^2{}_G\,\alpha$). Then $\forall i \in G$. $B_i(\alpha \wedge E_G\,\alpha)$.

Assuming that agent *i* believes that the other agents *j* have the same reasoning system for beliefs as itself. Given that agent *i* believes that every agent is following the same protocols, then everything agent *i* believes has been achieved, it believes everyone believes so too from positive introspection. Therefore $\forall i, j \in G \,.\, B_i B_j(\alpha \wedge E_G\, \alpha)$ implying that $\forall i \in G \,.\, B_i\, \mathrm{E}_G\, (\alpha \wedge E_G\, \alpha)$, that is $E^3_G \alpha$.

Theorem2 depends on the consistency of beliefs of the member agents in a group. Since an interaction protocol defines only a finite number of states, then positive and negative introspection about the end states are possible and does not require complete beliefs. In our case, an agent's introspection concerns only beliefs about what is being communicated in a specific interaction and more particularly termination. In case of unsuccessful termination such as timeouts, then negative introspection allows all agents to believe that not all agents believe the end state. A safe protocol thus allows all agents to believe any failure that occurs.

Corollary If the terminal state α is shared belief in a group *G*, then an interaction terminates without uncertainty.

The corollary follows from the state of interaction being common belief, given the preconditions of theorem2. Thus one advantage of reasoning about beliefs instead of knowledge is the ability to attain common beliefs about the (terminal) state of an interaction. Common knowledge is more difficult to reach since what an agent knows would have to be true (rather than merely consistent as in the case of common belief).

7.11 Definitions and Assumptions

In a perfect communication medium, where message exchange is error-free and instantaneous, belief revision after state transition messages can be instantaneous. In this case, the synchronisation protocol is straightforward. However this is not the case for time-constrained interactions in unreliable networks. To this end, we specify synchronisation protocols between two interacting agents to ensure their interaction progresses only with consistent joint beliefs about the interaction state. Our protocol allows a sender to eventually believe whether others have successfully received its message and receivers to believe whether their acknowledgments have been received.

This enables each agent to believe the same interaction state before, during and after each state transition with an increasing degree of shared belief.

Several characteristics of a communication medium influence the design and choice of a synchronisation protocol. The interaction itself may be time-constrained or all the participants may benefit from unbounded time.

- *Perfect communication*.
  Message exchange is trivially delayed, error-free, not duplicated and never lost.

- *Imperfect communication but guaranteed receipt of message*.
  The network may delay, duplicate or corrupt messages but does not fail permanently so that a repeatedly sent message eventually reaches its destination. On not receiving an expected message, a receiver believes that either the message will eventually arrive or that the sender has crashed. A protocol using repeated messaging is suitable when the participants have unbounded time or their deadline is far. However such protocols increase the risk of network bottlenecks.

- *Non-guaranteed receipt of messages*.
  In this case, a message may never get relayed. Non-receipt of an expected message can be for any reason where a sender or the network has failed.

A family of protocols are possible for synchronisation in bilateral and multi-lateral interactions depending on the complexity of solutions an agent chooses in dealing with communication failures. We provide synchronisation protocols for guaranteed and non-guaranteed receipt of messages in a bilateral interaction. These protocols trivially apply when communication is perfect.

7.11.1 Assumptions

This section provides reasonable assumptions in order for the synchronisation protocols to have the desired effect of ensuring the safety of an interaction. These assumptions are the foundations of our reasoning and proofs.

Assumption1 The protocols (interaction, synchronisation and communication) are common beliefs. Before an interaction, all participants have agreed on which protocol to comply with. For example a market-place or an auctioneer broadcasts the types of auction it supports.

Assumption2 Following assumption1, an interaction is initiated in a commonly believed state. For example, in an auction if *open* and *closed* states are sub-states of the overall parent state *auctioning*, then the state of the auction before its start is ¬*auctioning*.

Assumption3 To avoid infinite acknowledgments in our protocol, we assume that the communication layer informs a sender if it fails to deliver its message (usually 100% packet loss can be detected).

Assumption4 All agents have perfect recall. An agent does not lose information erroneously about its beliefs, resulting in persistency of the states in an interaction.

7.11.2 Acknowledgments and Message Structure

We adopt a tuple for the structure for a message: (*message-number, sender, receiver, action, new-state*). A *message-number* associated with the sender avoids confusion in case of messages arriving in the wrong order or ghost messages. The fields *sender* and *receiver* are the identities of the sending and receiving agents respectively. A hash value can be used to detect whether a message has been corrupted by the network. The parameter *action* is a process from the sender for transition to the state *new-state* according to an interaction protocol. An *action* may be a FIPA-ACL or KQML performative. Unlike standard approaches to agent communication where belief revision is specified at a higher level than the message structure, here the state of the interaction is part of the message. The first reason for this is that a protocol may allow an action to trigger two different states, especially with the nesting of interactions. Another reason is that the *action* may be a complex process involving several consecutive state changes.

We define two types of acknowledgment to a message: *implicit* and *explicit*. An *implicit acknowledgment* contains a performative for a state change, whereas an

explicit acknowledgment is just an acknowledgment without any performative. An explicit acknowledgment includes an *ack* keyword as action instead of a performative. Let an agent *a* send a message, *m1*, containing a performative *offer* and a message number *i*, to agent *b* according to an interaction protocol \textit P. If protocol *P* specifies that receiver *b* has to respond with a performative (*agree* or *reject*), then the next message, *m2*, from *b* contains one of these two performatives. Message *m2* is also an implicit acknowledgment that *b* has received message *m1*. When *a* receives *m2* and the message number is (*i*+1), it can believe that *b* has received *m1*. Thus *m2* is said to be an implicit acknowledgment for *m1*. Implicit acknowledgements reduce the redundancy occurring in protocols where only explicit acknowledgements can be sent after each message. Implicit acknowledgments are not a new concept but constitute another reason for having the state as a parameter in a message.

When sending implicit acknowledgements and messages, we propose that an agent takes an *initiator* role – *initiator*(*X*) holds if the protocol allows *X* to send the next action causing a state transition. An agent chooses a role if the protocol allows it and according to its strategy for achieving its goals. Our synchronisation protocols can support (but do not enforce) the agents adopting new roles dynamically where an agent can implicitly be a sender or a receiver of a message. Therefore, an agent *X* can send a sequence of messages if the condition *initiator(X)* holds during that sequence (as when continually bidding a higher price in auctions) or it can decide to be a receiver.

If a receiving agent sends the wrong message back as in an incorrect acknowledgement, then the latter is ignored and the sending agent repeatedly sends its message as in the case of protocol *R* in section 7.12.

7.11.3 Terminating an Interaction

We specify that an interaction should be terminated by exchanging *four* consecutive explicit acknowledgments for achieving shared beliefs about the final state. The sender of the last performative leading to the terminal state sends the last (fourth) acknowledgment. Two acknowledgements ensure that all agents believe ($\alpha \wedge E_G \alpha$) and four acknowledgments result in all agents believing ($\alpha \wedge E_G \alpha \wedge E^2_G \alpha$), where α

is the terminal state. Thus four acknowledgments force clean termination of an interaction with shared beliefs about the final interaction state.

For the purposes of presentation in this paper, synchronisation protocols will be expressed as statecharts where separate states are propositions and arcs represent processes. The abbreviations $[X\backslash Y]$ and $B_X s_i$ are explained below.

7.12 Guaranteed Receipt of Messages

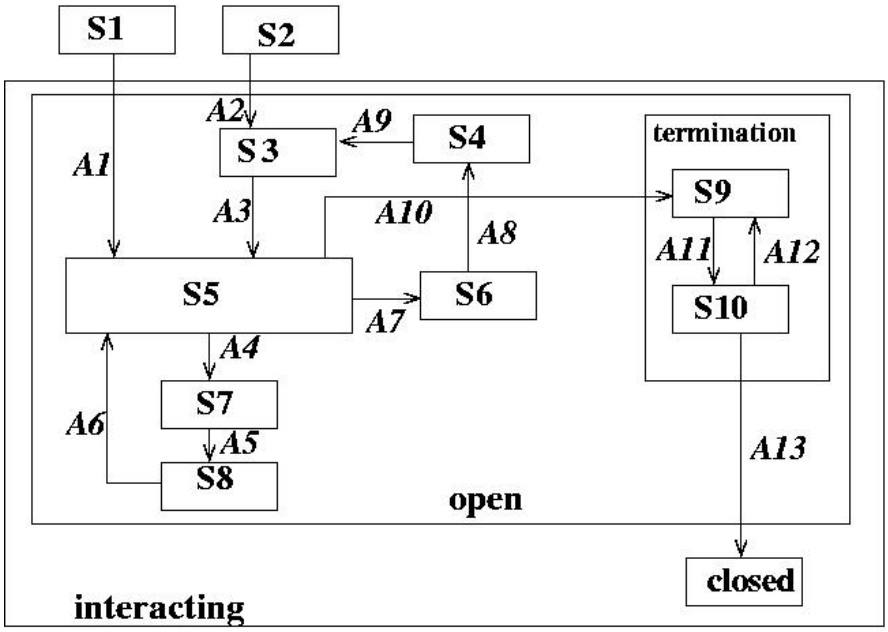


Figure 7.5 Synchronisation Protocol $\mathcal{R}$

S1	$\neg initiator(Y)$, $B_Y \neg interacting$
S2	$initiator(X)$, $B_X s_{i-1}$, $(s_{i-1} \leftrightarrow \neg interacting)$
S3	$mesg\_sent(X)$, $B_X s_{i-1}$
S4	$expl\_ack\_received(X)$, $B_X s_i$, $B_X B_Y s_{i-1}$, $B_X B_Y B_X s_{i-1}$
S5	$mesg\_received\ (Y)$, $B_Y s_i$
S6	$expl\_ack\_sent(Y)$, $B_Y s_i$
S7	$impl\_ack\_sent(Y)$, $B_Y s_i$
S8	$impl\_ack\_received(X)$, $B_X s_{i+1}$, $B_X B_Y s_i$, $B_X B_Y B_X s_{i-1}$
S9	$sent\_ack_m\ (Y)$, $\neg E^2_G s_t$, $B_Y E^j_G s_t$
S10	$received\_ack_m(X)$, $B_X E^{j+1}_G s_t$

A1	$Y.receive(i, X, Y, a_i, s_i)$
A2	$(B_X B_Y s_i ?; X.send(i, X, Y, a_i, s_i))^+$
A3	$Y.receive(i, X, Y, a_i, s_i)$
A4	$initiator(Y)?; (B_Y B_X s_{i+1} ?; Y.send(i+1, Y, X, a_{i+1}, s_{i+1}))^+$
A5	$X.receive(i+1, Y, X, a_{i+1}, s_{i+1})$
A6	$[X\backslash Y]; [i \setminus (i+1)]$
A7	$(B_Y B_X B_Y s_i ?; Y.send(i, Y, X, ack, s_i))^+$
A8	$X.receive(i, Y, X, ack, s_i)$
A9	$(B_X B_Y s_{i+1} ?; X.send(i+1, X, Y, a_{i+1}, s_{i+1}))^+ ; [Bxs_{i-1} \setminus B_X s_i]$
A10	$(s_i \leftrightarrow s_t)? ; (B_Y B_X B_Y s_t ?; Y.send(i, Y, X, ack_1, s_t))^+ ; [ack_m \backslash ack_1] ; [j \setminus 0]$
A11	$X.receive(m, Y, X, ack_m, s_t)$
A12	$(\neg E^2_G s_t)?; ((B_X E^{j+1}_G s_t)?; X.send(m, X, Y, ack_{i+1}, s_t))^+ ;$
A13	$(E^2_G s_t)?$

Table 7.1 The states and processes of protocol $\mathcal{R}$

In this and the next section, we specify and validate synchronisation protocols between two agents for systems in which there is guaranteed (Figure 7.5, protocol $\mathcal{R}$) and non-guaranteed (Figure 7.7, protocols $\mathcal{J}$ and ($\mathcal{R}$+$\mathcal{J}$)) receipt of messages. Unlike the protocols in [Fagin and al. 1995], the sender and receiver roles are not reserved to one agent. We allow agents to change their roles dynamically because any agent may send a message in an interaction according to the common interaction protocol. This is signified by the meta-process $[X\backslash Y]$ where Y replaces X (and vice versa). In an interaction, a sender of a message may become the receiver of the next message. Therefore we cannot separate the synchronisation protocol of the sender from that of the receiver (as do [Fagin and al. 1995]). This, in turn, increases the complexity of the synchronisation protocols.

The basic idea behind our synchronisation protocols is that a sender does not revise its beliefs with the state transition it is proposing, but rather waits for an acknowledgement of the receipt of its message before doing so. A sender can predict the next interaction state and the group's future beliefs, provided its message is successfully received, of which it can only be sure when receiving an

acknowledgement. As an interaction progresses with acknowledgements, an agent's predictions are discharged while increasing the level of mutual beliefs. In what follows, we analyse the case of guaranteed receipt of a message, when even though communication is imperfect, the network has not failed entirely.

The synchronisation protocol, called $\mathcal{R}$ in Figure 7.5 compensates for message delays and loss by repeatedly sending a message with the assumption that it is eventually delivered and the agents update their beliefs about the interaction state. Protocol $\mathcal{R}$ considers an interaction between two agents X and Y. Let the states of the bilateral interaction follow the sequence $s_0 \ldots s_i \ldots s_n$ and let s_t denote a terminal state where there are no actions possible from s_t for progress of the same interaction instance. The formula $B_X s_i$ is used to express the proposition that X believes the state is s_i. The states SM, $(1 \leq M \leq 10)$, and actions AM, $(1 \leq A \leq 13)$, in Figure 7.5 and the protocol $\mathcal{R}$ are defined in Table 7.1. The synchronisation protocol can be fully specified in ANML but here for the sake of conciseness we provide the states and actions in the protocol.

In the synchronisation protocol $\mathcal{R}$ if agent X is the sender of a message, then it keeps sending its message until it believes that agent Y has received it. As a sender, X does not yet believe the new state is s_i which it proposes, until it receives Y's acknowledgement. On receiving an explicit acknowledgement, X then believes the state is s_i along with Y's beliefs about both X and s_i. If instead X receives an implicit acknowledgement with an action to change to state s_{i+1}, X first believes the state s_i, that it previously proposed, has been reached, then it updates its beliefs to accept s_{i+1} as the new state and whatever it believes Y believes. Thus, X revises its belief about the interaction state consecutively twice from s_{i-1} to s_{i+1}. After receipt of an implicit acknowledgement, if X is not the initiator, it keeps sending an explicit acknowledgement until it believes Y has received it. At this stage, X and Y may switch roles. If X is the initiator after either an implicit or explicit acknowledgement, X keeps sending the next state transition in the interaction protocol until it receives another acknowledgement. Termination of the interaction is ensured with a final terminating loop by repeatedly exchanging explicit acknowledgements until each agent believes $(E^2_G s_t)$, where the terminal state s_t holds in the *closed* state.

7.13 Proof of Safety of Protocol $\mathcal{R}$

Safety here means: (i) there is no inconsistency in the joint beliefs of the agents about the interaction state before and after each state transition, (ii) that an interaction terminates in a state that is shared belief between the two agents. This section proves that protocol $\mathcal{R}$ exhibits both these features.

7.13.1 Proof of Consistency of Joint Belief

Simultaneous changes in beliefs at both agents' sides about the interaction state are unfeasible because messages do not get relayed instantaneously. Given this, our work aims to ensure there is a point where all agents have similar beliefs about the state of an interaction before the next state transition. We prove by induction that the synchronisation protocol $\mathcal{R}$ ensures attaining consistency of joint beliefs between agents *X* and *Y* after each state-triggering message (a message containing an action for a state transition). To do so, we need to prove that after each state-triggering message to s_i, both agents individually come to believe s_i , resulting in $(E^2_G\, s_i)$, before the next state transition.

Base Case: From Assumption2, at the start of the interaction the interaction protocol and the state of interaction are common belief.

Induction Hypothesis: At point *k*, assume agents *X* and *Y* believe the interaction state is s_1.

Induction Proof: At point (*k+1*), (e.g. *S2*, *A2* in Figure 7.1), let agent *X* keep sending a message m_2 for a state transition to s_2 until $B_X B_Y s_2$. When agent *Y* receives m_2, (e.g. state *S5* in Figure 7.5), it starts believing the state is s_2 and repeatedly sends an implicit or explicit acknowledgement for s_2 at point (*k+2*), (e.g. processes *A4*, *A7*). Agent *X* still believes the state to be s_1. At point (k+3), (e.g. state *S4* or *S8*), on receiving either an explicit or implicit acknowledgement, agent *X* updates its belief to s_2 and stops sending m_2. Both agents believe the state is s_2 at point (*k*+3). If the acknowledgement was implicit, agent *X* then updates its beliefs to the next state s_3 .

From this, a receiver believes that the sender has not changed its beliefs about the interaction state, but the receiver can believe that the sender is predicting a change. Eventually each agent believes that both agents previously believed s_2. Reasoning about the previous beliefs of an agent is possible through an interaction protocol, and helps to remove uncertainty about the history of the interaction.

7.13.2 Proof of Termination with Shared Belief

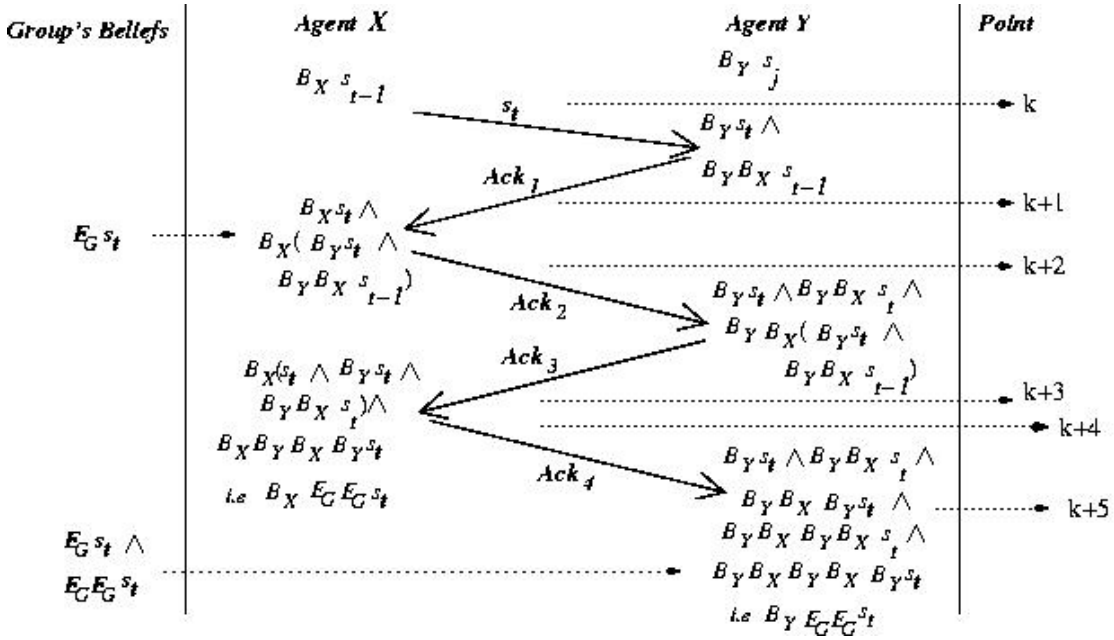


Figure 7.6 Termination with shared beliefs

To end an interaction, *X* and *Y* send four acknowledgements between themselves, for the group and each agent to believe ($E^2_G\, s_t$) where s_t is the terminal state (see Figure 7.6).

At point k. Agent *X* repeatedly sends a message for closing the interaction in state s_t until $B_X\, B_Y\, s_t$. *X* believes a state prior to s_t. *Y* receives *X*'s message, updates its belief to s_t and predicts $E_G\, s_t$.

Point (k+1): first acknowledgement Ack_1. *Y* believes s_t and repeatedly sends Ack_1 until $B_X\, B_Y\, s_t$, for letting *X* believe that *Y* believes s_t.

Point (k+2): second acknowledgement Ack_2. X receives Y's acknowledgement Ack_1, stops sending s_t and revises its belief to $(B_X\, s_t \wedge B_X\, B_Y\, s_t)$. At this point everyone believes s_t (i.e. $E_G\, s_t$). X repeatedly sends Ack_2 until it receives Ack_3 from Y.

Point (k+3), third acknowledgement Ack_3. Y receives Ack_2 and stops sending Ack_1. Y adds to its belief $B_X\, B_Y\, s_t$ and sends Ack_3 until it receives Ack_4 from X. Y does not believe $E^2_G\, s_t$ yet because it cannot be sure that X has received Ack_3.

Point (k+4), fourth acknowledgement Ack_4. X receives Y's acknowledgement Ack_3, stops sending Ack_2 and revises its belief to $(B_X\, s_t \wedge B_X\, B_Y\, s_t \wedge B_X\, B_Y\, B_X\, B_Y\, s_t)$. X believes $(s_t \wedge E_G\, s_t \wedge E^2_G\, s_t)$. X repeatedly sends Ack_4.

Point (k+5), Y receives Ack_4. Y adds $B_Y\, B_X\, B_Y\, B_X\, s_t$ to its belief i.e. $B_Y\, E^2_G\, s_t$. Thus each agent now believes the terminal state and shared beliefs about s_t – $(s_t \wedge E_G\, s_t \wedge E^2_G\, s_t)$.

7.14 Non-Guaranteed Receipt of Messages

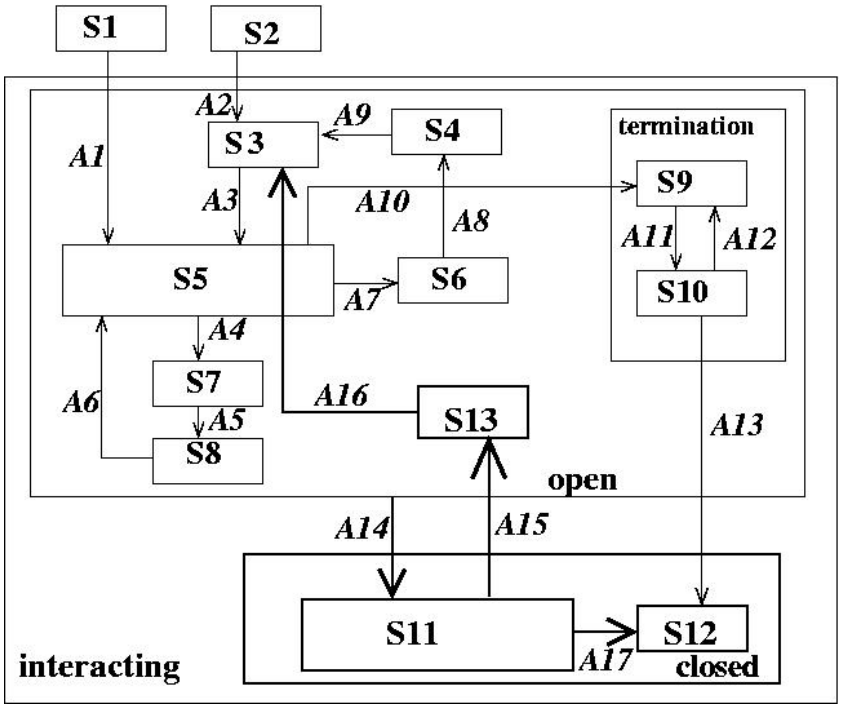


Figure 7.7 Protocols $\mathcal{T}$ and $(\mathcal{R}+\mathcal{T})$ for non-guaranteed receipt of messages

Usually there is no guarantee that a message will eventually reach its destination. In such cases, according to assumption 3, a sender is informed if its message fails to be delivered. On the other hand, a receiver *X*, which is expecting a message that does not arrive, cannot know whether the communication layer is at fault or the other agent *Y* is failing to respond because it has crashed. To this end, we specify a simple synchronisation protocol $\mathcal{T}$, (Figure 7.7), based on timeouts, where an agent waits for a message for a finite time *t*. Let the interaction state *timedout* hold after a *timeout*. Unlike for other interaction states *S1* to *S10*), given the failure of messages in reaching their destination, it is not possible to send four acknowledgements between the agents in order to achieve the shared belief E_G^2 *timedout*. Therefore, shared beliefs about timeouts are reached through negative introspection; every agent believes that every agent believes that the interaction state is unknown and therefore *timedout*.

In protocol $\mathcal{T}$, messages and acknowledgements are not repeatedly sent as in protocol $\mathcal{R}$. Therefore for protocol $\mathcal{T}$, the actions *A1* to *A13* in Figure 7.7 do not have the test conditions and repeated sending of Table 7.1. The states and actions (processes) in Table 7.1 also holds for protocol $\mathcal{T}$ after removing the test conditions and iterations from sending messages (in processes *A1* to *A13*) in protocol $\mathcal{R}$. The new states in Figure 7.7 for protocols $\mathcal{T}$ are:

S11	*received_timeout*(*X*), B_X *timedout*
S12	$E^2_G\, s_t \vee E^2_G$ *timedout*
S13	*restarted*(*X*), $E_G\, s_i$
A14	*X.receive*(*i*, _ , *X, timeout ,received_timeout(X)*)
A15	*X.send* (*n, X, Y, restart* ,s_i)
A16	[$Bxs_{i-1} \setminus E_G\, s_i$]
A17	ε

If a sender *X* is informed of failure of delivery of its message, then it believes the interaction has failed in the *timedout* state. The same applies for termination of an interaction through four acknowledgements when the sender of an acknowledgement is informed that its acknowledgement cannot be delivered. Similarly, if an agent *X* as

a receiver is waiting for a message which does not arrive before a timeout, then it believes the interaction state to be *timedout*. If an agent believes a timeout (state *S11*), it can restart (action *A15*) the interaction to the previous state that is commonly believed (leading to *S13*).

7.14.1 Proof of Safety of Protocol $\mathcal{T}$

As for protocol $\mathcal{R}$, two proofs are required – attaining consistency in the joint beliefs during an interaction and proof of safely terminating an interaction.

Consistency Proof. The proof for belief revision after a message or acknowledgement is similar to that of protocol $\mathcal{R}$ when the messages are delivered. A sender does not update its beliefs about the message it sends, but a receiver updates its beliefs on receiving explicit or implicit acknowledgements. In the case of timeouts and restarts, consistency is ensured by restarting the interaction in a previous state that is believed by everyone.

Termination Proof. If there are no timeouts, the proof for safe termination for protocol $\mathcal{R}$ also applies to protocol $\mathcal{T}$. If a timeout occurs, there are two cases for the termination proof depending on the role of an agent. First, if *X* is a sender and believes a *timedout* state from being notified by the network, from negative introspection the receiver *Y* also eventually believes a *timedout* state on not receiving *X*'s message or acknowledgement. Second, if *X* is a receiver and is the first to believe a *timedout* state, then the sender *Y* on not receiving an acknowledgement eventually believes the *timedout* state from negative introspection. From the property of negative introspection, all agents believe they are not aware what is the successful end state of an interaction. So they all believe that they all believe that not all agents believe the interaction is successful. Similarly if a timeout occurs during the exchange of the first 3 acknowledgements of a terminal state, then from negative introspection either agent will come to believe the *timedout* state because they will not receive the next acknowledgement. Regarding the fourth acknowledgement ack_4, firstly if it cannot be delivered by the network then the sender will be notified and the receiver will not receive ack_4. If the sender has crashed, then the receiver does not receive ack_4. From

negative introspection, whenever an agent does not receive ack_4, it eventually believes *timedout*.

7.15 Repeated Messaging and Timeouts

Protocols $\mathcal{R}$ and $\mathcal{T}$ can be combined to give the hybrid protocol ($\mathcal{R}$+$\mathcal{T}$) for more flexible behaviour in cases where there is both guaranteed and non-guaranteed receipt of messages. Protocol ($\mathcal{R}$+$\mathcal{T}$) includes the timeouts from protocol $\mathcal{T}$ and, from protocol $\mathcal{R}$ the ability for an agent to keep sending a message or an acknowledgement until it believes that its message or acknowledgement has been received.

Thus, an agent repeatedly sends a message until it receives an acknowledgement or a timeout, in which case it restarts the interaction in a commonly believed previous state. For the protocol ($\mathcal{R}$+$\mathcal{T}$) in Figure 7.7, actions *A1* to *A13* are similar to those in Table 7.1 for protocol $\mathcal{R}$. The states *S11*, *S12*, *S13* and actions *A14*, *A16*, *A17* are the same for both protocols $\mathcal{T}$ and ($\mathcal{R}$+$\mathcal{T}$). The action *A15* in protocol $\mathcal{T}$ is replaced by $((B_X B_Y\, restarted)?; X.send\ (n, X, Y, restart, s_i))^*$ for protocol ($\mathcal{R}$+$\mathcal{T}$).

When the communication framework guarantees delivery of a message, then the hybrid protocol ($\mathcal{R}$+$\mathcal{T}$) can be used if the agents have deadlines. If an agent reaches its deadline while waiting for a message, it may choose to timeout (especially when the network is lagging). Realistically, an agent cannot keep sending a message for periods extending over days or weeks without the risk of network bottlenecks or wasting computation time and power. In these cases or in case of network lags, timeouts are needed to halt the interaction process. In the non-guaranteed case, repeated sending of a message increases the probability of it reaching the destination. Thus protocol ($\mathcal{R}$+$\mathcal{T}$) combines the advantages of both protocols $\mathcal{R}$ and $\mathcal{T}$ for realistically coping with broader reasons for failures instead of specific defects in the communication medium.

Proof Both the safety proofs for protocol ($\mathcal{R}$+$\mathcal{T}$) in reaching consistency in the joint beliefs of the group about an interaction state and the safe termination of an

interaction follow from the proofs for protocols $\mathcal{R}$ and $\mathcal{T}$. Protocol ($\mathcal{R}$+$\mathcal{T}$) inherits the properties of its parent protocols $\mathcal{R}$ and $\mathcal{T}$.

7.16 Pragmatics of Synchronisation

Dealing with the consequences of communication failures in agent interaction has usually either been relegated to the ISO transport layer or has been analysed through knowledge-based reasoning about the transmission of bits over a network. This chapter argues that agents need to reason about and compensate for message delays and loss to avoid contradictory beliefs about the interaction state. Thus, a belief-based approach is adopted for achieving a degree of common beliefs amongst a group of agents about an interaction. Reaching an equivalent degree of knowledge is more complex (if not impossible) given the property of knowledge being true. Specifically, we ensure and prove the safe progress and termination of an interaction through synchronisation protocols given an agent is capable of positive and negative introspection over terminal states.

This chapter argues that synchronisation protocols take into account the realistic aspects of agent interaction where the communication network may influence the setting, especially in the area of wireless telecommunications. The safety of the protocols in reaching consistency and for safe termination depends on the protocols being complete. This may be a strong assumption in cases where protocols are designed arbitrarily. However even in open environments with dynamic entry and departure of agents, the interaction protocol is fixed at the beginning of the negotiation. The properties of the protocol and in particular completeness can be ensured by its providers or the institution supporting the trading. Given that only a finite set of states and actions are defined by a protocol, then checking for completeness is not intractable and can be facilitated or automated by techniques such model checking.

7.17 Fragment of Bilateral Negotiation

The synchronisation protocol is applied to a bilateral negotiation between two agents *X* and *Y*. Table 7.2 is a fragment of a bilateral negotiation from a *proposed*(*Y*) state to an *agreed* state showing the beliefs of each agent.

Event	**Neg. state believed by *X* and *Y* privately before the event i.e. B_X and B_Y**	**Neg. state believed by *X* and *Y* privately after the event i.e. B_X and B_Y**	**Action performed after the event**
…	...	...	...
X sends a *request*, *X.request*	B_X *proposed*(*Y*) B_Y	B_X *proposed*(*Y*) B_Y	*X* waits for an acknowledgement
Y receives *X*'s *request*	B_X *proposed*(*Y*) B_Y	B_X *proposed*(*Y*) B_Y *requested*(*X*)	*Y* sends an *offer*
Y sends an *offer*, *Y.offer*	B_X *proposed*(*Y*) B_Y *requested*(*X*)	B_X *proposed*(*Y*) B_Y *requested*(*X*)	*Y* waits for an acknowledgement
X receives and first deals with *Y's* implicit acknowledgement	B_X *proposed*(*Y*) B_Y *requested*(*X*)	***B_X requested*(*X*)*** ***B_Y requested*(*X*)***	*X* deals with *Y* 's *offer*
X deals with *Y*'s *offer*	***B_X requested*(*X*)*** ***B_Y requested*(*X*)***	B_X *offered*(*Y*), B_Y *requested*(*X*)	*X* sends an *agree*
Y receives *X*'s message and deals with its implicit acknowledgement.	B_X *offered*(*Y*) B_Y *requested*(*X*)	***B_X offered*(*Y*)*** ***B_Y offered*(*Y*)***	*Y* deals with *X's* agreement
Y deals with *X's* agreement	***B_X offered*(*Y*)*** ***B_Y offered*(*Y*)***	B_X *offered*(*Y*) B_Y *agreed*	*Y* sends an acknowledgement, A_1
...	…	...	...
after 4th acknowledgement from *X*	***B_X agreed*** ***B_Y agreed***	***B_X agreed*** ***B_Y agreed*** (Bel_G *agreed* $\wedge$ $Bel^2{}_G$ *agreed*	negotiation has terminated

Table 7.2 Fragment of a bilateral negotiation showing termination

7.18 Generic Scenario of a Negotiation

This section describes a simple and generic scenario of a negotiation between two agents, *X* and *Y*, to illustrate state transitions from an initial to a terminal state and the attainment of shared beliefs in the group. A prediction operator is used in addition to the modal belief operator. $P_j\,\alpha$ indicates agent *j* predicts α. Group *G* is the group of agents *X* and *Y* i.e. $\{X,Y\}$. A prediction operator is used because an agent can predict α given the message for making α hold is successfully received.

7.18.1 Interaction Protocol in Scenario

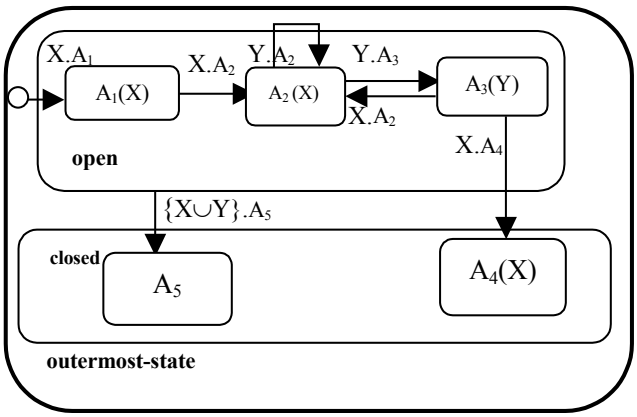


Figure 7.8 A generic scenario in state chart

Let two agents *X* and *Y* negotiate according to Figure 7.8. A logical theory for the interaction protocol is given in ANML below:

outermost-state ↔ one-of({ *open*, *closed*})

open ↔ one-of({ A_1 , A_2 , A_3 })

closed ↔ one-of({ A_5, A_4 })

¬*outermost-state* ↔ none-of({ *open*, *closed*})

¬*open* ↔ none-of({ A_1 , A_2 , A_3 })

¬*closed* ↔ none-of({ A_5, A_4 })

¬*outermost-state* ↔ [$X.A_1$] $A_1(X)$

$A_1(X) \leftrightarrow [X.A_2]\ A_2(X)$

$A_2(X) \leftrightarrow ([Y.A_2]\ A_2(Y) \vee [Y.A_3]\ A_3(Y)) \wedge \neg(X = Y)$

$A_3(X) \leftrightarrow ([Y.A_2]\ A_2(Y) \vee [Y.A_4]\ A_4(Y)) \wedge \neg(X = Y)$

$open \leftrightarrow ([X.A_5]\ A_5 \vee [A_3(X)?;\ Y.A_4]\ A_4(Y)) \wedge \neg(X = Y)$

Theory 7.1 Interaction Protocol of Scenario in ANML

7.18.2 A Possible Path of Negotiation

From the assumptions, agents X and Y commonly believe the state at the start of the negotiation to be ¬(*outermost-state*). Table 7.3 illustrates a possible negotiation following the interaction protocol in Theory 7.1 and the synchronisation protocol. The term x_n denotes the state agent X believes at point n in the sequence of sent performatives. Likewise for the state y_n believed by agent Y.

Agent X's beliefs, B_X	**Agent Y's beliefs, B_Y**
$x_0 \leftrightarrow \neg$ (*outermost-state*) By introspection: $B_X\, x_0\, y_0$ Common belief about x_0 and y_0: $B_X\, B_Y x_0\, y_0 \wedge B_X\, B_Y\, B_X\, x_0\ y_0 \wedge \ldots\ldots$ Possible state transitions: $B_X\,(x_1\ y_1 \leftrightarrow A_1(X) \vee A_1(Y))$ $B_X\, B_Y\,(x_1\ y_1 \leftrightarrow A_1(X) \vee A_1(Y))$	$y_0 \leftrightarrow \neg$(*outermost-state*) Introspection: $B_Y\, x_0\, y_0$ $B_Y\, B_X x_0 y_0 \wedge B_Y\, B_X\, B_Y\, x_0\, y_0 \wedge \ldots\ldots$ Possible state transitions: $B_Y\,(A_1(X) \vee A_1(Y))$ $B_Y\, P_X\, A_1(X) \vee A_1(Y)$
X repeatedly sends (*1*, X, Y, $X.A_1$, $A_1(X)$, *hash value*)	
$x_0 \leftrightarrow \neg$ (*outermost-state*) Common belief about x_0 and y_0: $B_X\, B_Y x_0\, y_0 \wedge B_X\, B_Y\, B_X\, x_0\ y_0 \wedge \ldots\ldots$ Possible state transitions: $B_X\,(x_1\ y_1 \leftrightarrow A_1(X) \vee A_1(Y))$ Predictions of X: $P_X\,(x_1\ y_1 \leftrightarrow A_1(X))$ Predictions about introspection: $P_X\, B_X\,(x_1\ y_1 \leftrightarrow A_1(X))$, $P_X B_Y\,(x_1\ y_1 \leftrightarrow A_1(X))$ X can even predict Y's predictions e.g. $P_z\, P_Y x_1$ and $P_z\, P_Y\, B_Y\, B_X\, x_1$ where $z = X$ or Y	Y receives message (*1*, X, Y, $X.A_1$, $A_1(X)$, *hash value*) from X Y changes its belief to $(y_1 \leftrightarrow A_1(X))$ Y believes the state at X: $B_Y\, B_X\, x_0$ and $B_Y\, y_1$ Common belief about x_0: $B_Y\, B_X x_0\, y_0 \wedge B_Y\, B_X\, B_Y\, x_0\, y_0 \wedge \ldots\ldots$ Y believes all that X predicts: $B_Y\, P_X\,(x_1\, y_1 \leftrightarrow A_1(X))$
According to the protocol, Y cannot perform the next state transition, so Y keeps sending an explicit acknowledgement: (*2*, Y, X, ack, $A_1(X)$, *hash value*)	
X receives Y's acknowledgement (*2*, Y, X, ack_a,	$y_1 \leftrightarrow A_1(X)$

$A_1(X)$, *hash value*) and stops sending its previous message number *1* $x_1 \leftrightarrow A_1(X)$ $B_X(x_1\ y_1 \leftrightarrow A_1(X))$ *X* believes that *Y* believes it was in state x_0 but not in x_1 yet: $B_X B_Y x_0 y_1$ *X* believes *Y*'s predictions about *X* going to x_1	The belief at *Y* is similar to the above row for *Y*. *Y* predicts the state and belief about x_1 and y_1 to be $A_1(X)$ **Both *X* and *Y* believe the current state to be $x_1 - Bel_G\ A_1(X)$**
X repeatedly sends (*3*, *X*, *Y*, *X.*A_2, $A_2(X)$, *hash value*)	
$x_1 \leftrightarrow A_1(X)$ *X* has the same belief as in the previous row for *X*. *X* has added predictions about message *3* $P_X(x_2\ \ y_2 \leftrightarrow A_2(X))$ $P_z\ P_X(x_2\ \ y_2 \leftrightarrow A_2(X))$ *X* predicts that *Y* still believes *X* to be in state $x_1 \leftrightarrow A_1(X)$ $P_X B_Y(x_1 \leftrightarrow A_1(X))$	*Y* receives message *3*. Message *3* is an implicit acknowledgement to *Y*'s acknowledgement (ack_1) to message *1*. *Y* stops sending its previous acknowledgement ack_1. *Y* believes that *X* has obtained *Y*'s ack_1 and that *X* believes state x_1. *Y* discharges its predictions and deals with the performative in message *3*. $y_2 \leftrightarrow A_2(X)$ and from introspection $B_Y(y_2 \leftrightarrow A_2(X))$ *Y* believes the state at *X* is A_1. $B_Y(x_1 \leftrightarrow A_1(X))$ Shared belief about x_1: $B_Y B_X\ x_1\ y_1$
Y repeatedly sends the next message (*4*, *Y*, *X*, *Y.*A_3, $A_3(Y)$, *hash value*)	
X receives *Y*'s message (*4*, *Y*, *X*, *Y.*A_3, $A_3(Y)$, *hash value*) and from the implicit acknowledgement in message *4*, *X* stops sending message number *3* *X* updates its belief about the state from *Y*'s implicit acknowledgement. $B_X(x_2 \leftrightarrow A_2(X))$ **Both *X* and *Y* have the same belief about the state $A_2(X)$.** *X* believes about *Y*'s beliefs: $B_X(y_2 \leftrightarrow A_2(X))$ *X* deals with the performative of *Y*'s message *4* and changes its belief to $x_3 \leftrightarrow A_3(Y)$	*Y* believes state to be $y_2 \leftrightarrow A_2(X)$ *Y* has the same belief as in the previous row for *Y*. *Y* adds predictions if *X* receives *Y*'s message *4*. The predictions resemble those when *X* was the sender except for a leading P_Y instead of P_X

X believes *Y* 's predictions: $B_X P_Y (x_3, y_3 \leftrightarrow A_3(Y))$	
X keeps sending (5, *X*, *Y*, *X.A*$_4$, $A_4(X)$, *hash value*)	
X has the same belief as in the previous row. Predictions of *X* after sending message 5 include the state at *Y* and *Y*'s belief. $P_X(y_3 \leftrightarrow A_3(Y))$ from acknowledgement and $P_X(y_4 \leftrightarrow A_4(X))$ from the performative. *X* predicts the final state at *X* and *Y* Belief at *X* is $x_3 \leftrightarrow A_3(Y)$. *X* has sent an action for terminating the negotiation in the closed state $A_4(X)$.	*Y* receives message *5* from *X* and stops sending message *4*. *Y* deals with the implicit acknowledgement and updates to $y_3 \leftrightarrow A_3(Y)$ **Both *X* and *Y* have the same belief about the state $A_3(Y)$ at this point.** *Y* discharges its predictions about message *4*. *Y* deals with the transition triggered by the performative in message *5*, $X.A_4$ and changes belief to $y_4 \leftrightarrow A_4(X)$
Y keeps sending its first explicit acknowledgement to terminate the negotiation (6, *Y*, *X*, ack*1*, $A_4(X)$, *hash value*). *Y* predicts $x_4 \leftrightarrow A_4(X)$ to hold at *X* and $B_G B_G A_4(X)$ to become shared belief.	
X receives *Y*'s first acknowledgement in message *6* and stops sending message *5*. *X* updates its belief $x_4 \leftrightarrow A_4(X)$. **Both *X* and *Y* have the same belief about the state $A_4(X)$ at this point.** By introspection *X* believes: $B_X (x_4\, y_4 \leftrightarrow A_4(X))$ Everybody believes the state $A_4(X)$: $Bel_G\, A_4(X)$	*Y* has the same belief as in the previous row for *Y*. $B_Y (y_4 \leftrightarrow A_4(X))$ *Y* adds predictions about the belief of *X* and of the group if its acknowledgement *ack1* successfully reaches *X*.
Both agents are in the same terminal state and $Bel_G\, Bel_G\, A_4(X)$ is ensured. *X* sends acknowledgement *ack2* in (7, *X*, *Y*, ack*2*, $A_4(X)$, *hash value*)	
X has the same belief as in the previous row for *X*. *X* believes that both agents believe the state. $B_X B_Y\, A_4(X)$ *X* predicts the belief of *X* on receiving *ack2*. $P_X\, B_Y\, x_4$, $P_X B_Y B_X\, x_4\, y_4$	*Y* receives *ack2* in message *7* and stops sending message *6* *Y* believes that both agents believe the terminal state ($B_Y B_G\, x_4\, y_4$) and that both agents believe that its opponent believes that they believe the terminal state ($B_Y B_X B_G\, x_4\, y_4$). *Y* discharges some of its

	predictions depending on the receipt of message *6*.
Y sends *ack3* in (8, *Y*, *X*, *ack3*, $A_4(X)$, *hash value*)	
X receives *ack3* in message *8* and stops sending message *7*. $B_X B_Y B_X x_4 y_4$	*Y* has the same belief as in the previous row for *Y*. *Y* predicts shared belief and the belief at *X*: $P_Y B_X B_Y B_X x_4 y_4$
X sends *ack4* in (9, *X*, *Y*, *ack4*, $A_4(X)$, *hash value*)	
$B_X B_Y B_X (x_4 y_4 \leftrightarrow A_4(X))$ The belief at *X* is $A_4(X)$, $Bel_G A_4(X)$ and $Bel_G Bel_G A_4(X)$	*Y* receives message *9* and *ack4* and stops sending message *8* $B_Y B_X B_Y (x_4 y_4 \leftrightarrow A_4(X))$. The belief at *Y* is $A_4(X)$, $Bel_G A_4(X)$ and $Bel_G Bel_G A_4(X)$

Table 7.3 Possible negotiation between agent *X* and *Y*

7.19 Summary

This chapter takes into account the work on joint intention by Cohen and Levesque [1991], epistemic logic by Fagin and al. [1995] and Meyer and Van der Hoek [1995]. The knowledge of a group of agents is analysed to ensure consistency in the joint knowledge about the protocol. We discuss attaining shared beliefs about the negotiation state between agents in an imperfection communication medium. To do so, conventions and a synchronisation protocol are specified to ensure that all participants believe the same negotiation state. The synchronisation protocols can be formalised in ANML in the same way as interaction protocols and their properties proved. The actual sending and receiving of messages and acknowledgements are part of a synchronisation protocol and not of an interaction protocol.

As further work, a suite of synchronisation protocols for bilateral and multilateral interactions can be designed according to the constraints, required flexibility and properties of the agents and network. For example, the converse of protocol ($\mathcal{R}$+$\mathcal{J}$) would involve repeated sending of a message *after* a timeout, thereby avoiding flooding the network. Merging and extensions of these synchronisation protocols can provide additional flexibility depending on network or agent unreliability. Despite

being more complex, synchronisation protocols for multi-lateral interactions may also reuse the theorems in section 7.10 as basic principles and represent a challenging avenue for future work. We also aim to explore the relation between interaction and synchronisation protocols.

8 Practical Agents

8.1 Introduction

A number of techniques have been used to implement strategies in multi-agent systems, [Sandholm 1999, Faratin 2001, Yokoo and Ishida 1999, Jennings and al. 2001]. These techniques include heuristic search, decision making mechanisms and game theory. Game theory, [Rasmusen 1989], can be used to study protocols and strategies between self-interested agents, [Rosenchein and Zlotkin 1994]. In such systems, an agent decides on its best course of action in response to received messages. Traditional game theoretic models are unrealistic as they assume rationality and complete knowledge of details of the game and equilibrium calculations of other agents. Finding an optimal solution from all possible strategies is computationally intractable and the assumption that an agent knows the utilities of the other participants are unfeasible.

Decision theory, [Raiffa 1982], analyses different alternatives under uncertain conditions and unknown outcomes of an action, for maximising the obtained utility of a decision-maker. Decision theory, for example Markov Decision processes, suits well the dynamic nature of agent systems and assumes that the state of the world at any point is known. This chapter shows how to add strategies and planning to a protocol. A simulation of a bilateral negotiation is described using the strategies in Faratin and al., [1999], which specify one-step decision making algorithms. Faratin and al., [1999], uses a simple protocol of offers and counter-offers. We design an agent to respond with more choice than these two primitives through the bilateral protocol. The experiments and results from an

MSc. Project, [Alogogianni 2001] are included. This chapter also discusses how to use a protocol in ANML in order to infer paths and plans towards a goal.

8.2 Strategies for Single Actions

Faratin and al. [1999] specify an agent architecture for decisions and action mechanisms between non-coorperating agents. A negotiation subject may consist of a set of issues and values attributed to the issues. Rational behaviour entails maximising a value function about the set of issues. They describe two types of mechanisms – responsive and deliberative – for evaluating messages from other agents and for deciding on how to respond, (offer generation mechanisms). A responsive strategy is computationally less costly and faster than a deliberative one, which involves a more complex search of the solution space. They consider a negotiation between two agents *a* and *b* over a changeable set of issues *J*. An issue *j*, ($j \in J$), can take values between [min_j, max_j], which define the domain, D_j, of a quantitative issue. The domain of a qualitative issue is defined as an ordered set of possible values as in $D_j = <q_1, \ldots, q_1>$.

8.2.1 Evaluation Mechanisms

A message from an agent contains a negotiation subject which consists of attributes or issues represented as name-value pairs. Let the term $(x_{b\to a})^t$ denote the values associated to the issues, sent from agent *b* to *a* at time *t*. Evaluation of $(x_{b\to a})^t$ involves computing its score (or value) by summing the score of each issue. An agent *i* defines a scoring function, V_j^i, about an issue *j* where V_j^i: $D_j \to$ [0,1] binds the score agent *i* assigns to issue *j* to values between 0 and 1. Issues are independent and the scoring functions are monotonous for quantitative issues.

Consider an agent *a* receiving $(x_{b\to a})^t$ from *b* at time *t*, over a set of issues *J*, where ($x = (x[j_1], \ldots, x[j_i], \ldots, x[j_n])$ and $j_i \in J$. Agent *a* associates weight $\omega_{j_i}^a$ with issue j_i where the sum of weights of the issues in $(x_{b\to a})^t$ is 1. These weights can be changed since a set of issues is allowed to change during a negotiation. An agent rates the overall subject through a weighted linear additive scoring function:

$$V^a(x) = \sum_{1 \le i \le n} \omega^a_{j_i} \; V^a_{j_i}(x[j_i])$$

$$\text{and} \sum_{1 \le i \le n} \omega^a_{j_i} = 1 \text{ and } V^a_{j_i} : D_j \rightarrow [0,1].$$

These evaluation functions may be used in evaluating a set of issues from a participant or to generate offers.

8.2.2 Responsive Mechanisms

Responsive mechanisms allow reactive behaviours depending on environmental factors like time, behaviour and resources. A response is generated by linearly combining simple decay functions, called tactics, [Faratin and al. 1999]. Three types of tactics are used: time-dependent, resource-dependent and behaviour-dependent where each tactic gives values for issues using only that environmental factor.

In time-dependent tactics, an agent concedes more as the deadline for its negotiation approaches. In resource-dependent tactics, an agent will concede more as the level of the resources diminishes. In behavioural tactics, concessions are based on the concessions of the other agents. A strategy can use a mixture of these tactics with associated weights depending on the relative importance of time, resources and behaviour.

Time-dependent tactics

Let t^a_{max} denote a deadline for agent a. An offer from a is generated from a function $\alpha^a_j : T \rightarrow [0,1]$. At time 0 an agent does not concede and at the deadline an agent concedes to its reservation limit, $\alpha^a(t^a_{max}) = 1$. An agent a sending an offer x to agent b for issue j at time t is denoted by $(x_{a \rightarrow b})^t [j]$

$$(x_{a \rightarrow b})^t [j] = min^a_j + \alpha^a_j (t) (max^a_j - min^a_j) \text{ if } V^a_j \text{ is decreasing}$$
$$(x_{a \rightarrow b})^t [j] = min^a_j + (1 - \alpha^a_j (t)) (max^a_j - min^a_j) \text{ if } V^a_j \text{ is increasing}$$

Faratin and al. [1998] compute the initial actions on entering a negotiation by asking the owner of an agent to enter a constant k^a_j for each issue j. The initial

offer is modelled as a point in the interval of values for each issue. The interval of values for an issue, j, is obtained from the difference between the reservation values, (min_j, max_j), of an agent for that issue. Multiplying constant k_j^a with the size of the interval determines the value of issue j in the first offer. A wide range of time-dependent functions can be defined by varying the computation of $\alpha_j^a(t)$ where $\alpha^a(0) = k_j^a$ and $\alpha^a(t^a_{max}) = 1$. An exponential and a polynomial way to model α can be found in [Faratin and al. 1999]. In our simulation instead of asking a user to input k_j for an issue, a user inputs its initial value preferences for each issue and k_j^a is computed by agent, a, for issue j. The following function is used to calculate $\alpha_j^a(t)$ from k_j^a.

$$\alpha_j^a(t) = k_j^a + (1 - k_j^a) \times \frac{\min(t, t_{\max})}{t_{\max}}$$

Resource-dependent tactics

An agent concedes more as the level of its resources decreases, including time. The valuation $\alpha(r)$ may be derived by estimating the amount of a particular resource, r. The offer x can then be modelled as for the time-dependent tactics. For example, if dealing with time then α may be calculated as follows:

$\alpha_j^a(t) = e^{-(t\_max - t)}$ where t_max is the deadline for agent a.

$\alpha_j^a(t) = e^{-resource(t)}$ where the function $resource(t)$ measures the quantity of resource at time t.

Behaviour-dependent tactics

An agent determines its concessions based on the previous attitudes of other agents. There are different ways in which an agent imitates other agents, given in [Faratin and al. 1998]. In the relative tit-for-tat tactic, an agent reproduces in percentage terms the behaviour its opponent performed ($\delta \geq 1$) steps ago.

$$(x_{a \to b})^t[j] = \begin{cases} min_j^a & \text{if } P \leq min_j^a \\ max_j^a & \text{if } P > max_j^a \\ P & \text{otherwise} \end{cases}$$

The value *P* relates to how the aspect and degree an opponent's behaviour is proportionally imitated. In relative tit-for-tat, the behaviour of an opponent ($\delta \geq 1$) steps ago is reproduced in percentage terms where

$$P = \left[\frac{x_{b \to a}^{t_{n-2\delta}}[j]}{x_{b \to a}^{t_{n-2\delta+2}}[j]} \right] x_{a \to b}^{t_{n-1}}[j]$$

($\delta = 1$) is used for an imitative behaviour early in a negotiation. The values in a set of issues when using relative tit-for-tat tactics is generated as follows:

$$x_{a \to b}^{t_{n+1}}[j] = min(max(\frac{x_{b \to a}^{t_{n-2\delta}}[j]}{x_{b \to a}^{t_{n-2\delta+2}}[j]} \times x_{a \to b}^{t_{n-1}}[j], \min_j^a), \max_j^a)$$

Offer $x_{a \to b}^{t_{n+1}}[j]$ is within the range of agent *a*'s acceptable values and proportionally imitates agent *b*'s behaviour. Our algorithm uses relative tit-for-tat as the default behavioural tactic. If $x_{b \to a}^{t_{n-2\delta}}[j]$, $x_{b \to a}^{t_{n-2\delta+2}}[j]$ or $x_{a \to b}^{t_{n-1}}[j]$ are equal to zero, then an agent *a* uses the absolute tit-for-tat tactic.

In random absolute tit-for-tat, imitation is in absolute terms. If an opponent of agent *a* decreases its proposal by *d* amount then *a* responds with an increase by *d*. An agent increases or decreases its response by a random amount following its opponent last message. Agent *a* concedes or demands exactly as much as its opponent did ($\delta \geq 1$) steps ago. ($\delta = 1$) is used. The function that computes the response from agent *a* is given below and ensures that the generated set of issues is within *a*'s acceptable range. The random component of the tactic in [Faratin and al. 1998] is disregarded.

$$x_{a \to b}^{t_{n+1}}[j] = \min(\max(x_{a \to b}^{t_{n-1}}[j] + (x_{b \to a}^{t_{n-2\delta}}[j] - x_{b \to a}^{t_{n-2\delta+2}}[j] , \min_j^a), \max_j^a)$$

Another tactic is averaged tit-for-tat, where an agent computes the average of percentages of changes in a fragment of its opponents' history.

8.2.3 Deliberative Mechanisms

Responsive strategies are computationally less complex than deliberative mechanisms. Deliberative mechanisms in Faratin and al [1999] are strategies for generating new offers which have the same value for an agent but better for its opponents. Two such mechanisms include the *trade offs* and *issue set manipulations* strategies. In the latter mechanism, a non-empty set of issues is shared and partly modified to resolve conflicts. The set of issues consists of "core" issues which cannot be altered and non-core issues that can dynamically be changed. Agents can add or remove issues to the non-core issues subset for solutions to a negotiation. An agent decides how to combine additions and removals of issues so as to maximise the value of its response. Finding the optimal set of issues may be computationally expensive.

In trade-offs mechanisms, an agent lowers its score on some issues and asks more on other issues. The offer has the same value as a previous response for that agent but is more beneficial to its opponent. In a responsive mechanism, an agent, *a*, compares the score of a proposal from agent, *b* with the response it will generate next. In a deliberative mechanism, an agent compares the score of a received proposal with its own previous proposal. This decision mechanism involves searching for all possible offers with the same score as the previous offer and selection of the offer which is "closest" to an opponent's last offer, [Faratin and al. 1999]. The set and value of issues that produce the same utility as a previous offer lie on the same iso-value curve, [Raiffa 1982]. An agent is indifferent to all offers that lie on the same iso-value curve.

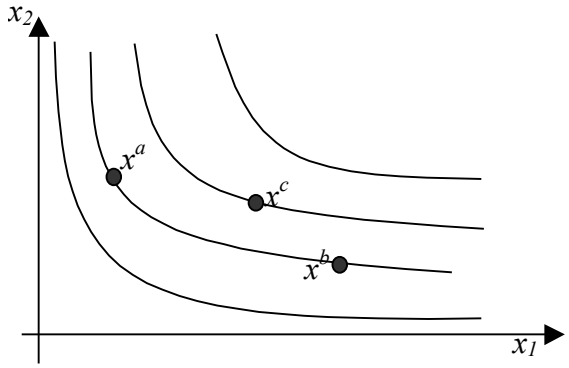


Figure 8.1 Iso curves

The above indifference curves show an agent's preferences over two issues x_1 and x_2. Given a score function for agent i, V^i, that agent associates the same value to points x^a and x^b i.e. $V^i(x^a) = V^i(x^a)$ and this equality applies to all the points on the same iso-curve. x^c is preferred to x^a or x^b and lies on an iso-curve with increased preference. Faratin and al., [1999], defines an iso-curve in terms of a scoring value θ. The iso-curve set at degree θ for an agent a is defined as:

$$iso_a(\theta) = \{x | V^a(x) = \theta\}$$

This gives the set of all responses that have the same value as the previous offer from agent a. A closeness function is then used to find the point (or offer) that would most resemble an opponent's last offer. The best trade-off would be the most similar response to an opponent on agent a's iso-curve. Given two consecutive offers, one from agent a to b, x, and one from b to a, y, the trade-off for agent a with respect to y is calculated as follows:

$$tradeoff_a(x, y) = average \max_{z \in iso_a(\theta)} \{Sim(z, y)\}$$

The scoring value of an offer x from agent a is θ i.e. $\theta = V^a(x)$. Hence $iso_a(\theta)$ gives the space of all possible set of issues which have the same value as $V^a(x)$. $z \in iso_a(\theta)$ denotes a possible set of issues, z, which have the same value as the set of issues x and max $\{Sim(z, y)\}$ returns the set z which is the closest to agent b's offer, y.

Similarity between two offers is defined as the weighted combination of the similarity of issues, [Faratin and al. 1999]. The similarity between two offers x and y over a set of issues J is calculated as:

$$Sim(x,y) = \sum_{j \in J} \omega_j^a Sim_j(x_j, y_j)$$

where $\sum_{j \in J} \omega_j^a = 1$. Sim_j is the similarity function for *issue j* and ω_j^a is the weight agent a gives to issue j. A similarity function, [Valverde 1985], can be defined as "a conjunction of appropriate fuzzy equivalence relations induced by a set of

criteria functions h_i". A criteria function is a function that maps values from a given domain into [0,1]. As an example, desserts are compared for modelling the similarity between them. The domain is $D_{desserts}$ = {*pie*, *pancake*, *icecream*, *tarts*, *cheesecake*, *sorbet*, *pudding*, *fruit*}. To compare the similarity between two elements from this given set of desserts, they are assigned different criteria such as temperature e.g. cold, warm, ordinary or calories, etc. These criteria can be used to obtain a set of name-value pairs where h_t and h_c are the comparison criteria functions.

h_t = {(*pie*, *0.9*), (*pancake*, *0.8*), (*icecream*, *0*), (*tarts*, *0.5*), (*cheesecake*, *0.3*), (*sorbet*, *0*), (*pudding*, *0.8*), (*fruit*, *0.5*)}
h_c = {(*pie*, *0.8*), (*pancake*, *0.3*), (*icecream*, *0.6*), (*tarts*, *0.7*), (*cheesecake*, *0.9*), (*sorbet*, *0.4*), (*pudding*, *0.8*), (*fruit*, *0.1*)}

For the function h_t, 1 denotes hot temperature and 0 denotes cold temperature. In the function h_c, 1 denotes maximum calories and 0 denotes minimum calories. Given a domain of values D_j, the similarity between two values, (or issues one in the offer and one lying on the same iso-space), $x_j, y_j \in D_j$ is:

$$Sim_j(x_j, y_j) = \min_{1 \le i \le m} (h_i(x_i) \leftrightarrow h_i(y_i))$$

where $\{h_1, \ldots, h_m\}$ is a set of comparison criteria and h_i: $D_j \rightarrow [0,1]$. $\leftrightarrow$ is an arbitrary equivalence operator. For the above example with desserts, $(h(x_i) \leftrightarrow h(y_i)) = (1 - |\, h(x_i) - h(y_i)|\,)$ is used as the equivalence operator. Another simple example of an equivalence operator is $(h(x_i) \leftrightarrow h(y_i)) = min(h(y)/\, h(x), h(x)/\, h(y))$. As example the similarity between desserts is calculated using the equivalence operator $(1 - |\, h(x_i) \leftrightarrow h(y_i)|\,)$.

$Sim_{desserts}$ (*icecream*, *sorbet*) = $min(1 - |\, h_t(icecream) - h_t(sorbet)|, 1 - |\, h_c(icecream) - h_c(sorbet)|\,) = min(1, 0.8) = 0.8$

$Sim_{desserts}$ (*icecream*, *pie*) = $min(1 - |\, h_t(icecream) - h_t(pie)|, 1 - |\, h_c(icecream) - h_c(pie)|\,) = min(0.1, 0.8) = 0.1$

This means from the given domain D_j, assigned criteria functions and equivalence operators, icecream is more similar to sorbet than to pie.

8.3 Strategies for a Bilateral Negotiation

In our simulation of a bilateral negotiation, the responsive and deliberative strategies for evaluating and generating a set of issues are combined with our strategies for deciding on state transitions. The negotiation subject consists of a set of issues as possible deals. When an agent receives a message containing a set of issues, it evaluates this set and decides either to terminate the interaction or to generate a response with a new set of issues according to its strategies. An agent sending a message can be regarded as the initiator of the current state and other agents that can trigger the next state are considered as respondents. Events such as timeouts can also occur. The bilateral protocol analysed in the previous chapters is used in addition to the capability of counter-offering and counter-proposing from a *proposed* state as in the following rule:

$$proposed(X)) \leftrightarrow [Y.request]\ requested(Y) \vee [Y.counter\text{-}propose]\ proposed(Y) \vee [Y.counter\text{-}offer]\ offered(Y) \wedge \neg(X{=}Y).$$

An issue is structured as a name-value pair where it may be a quantitative or qualitative issue. Issues are mutually independent and can be added or removed from a set or their values changed. The agents in a group may have opposing interests for these issues and negotiation is the process for finding a space of mutual interest or satisfaction of the participants' goals. [Jennings and al. 2000] defines a negotiation as a distributed search through a space of potential agreements made of possibly acceptable sets of issues. When new issues are added, extra dimensions are added to the negotiation space leading to an increase in the number of points of agreements and vice versa. Changing the values of an issue leads to a different point in the agreement space.

A bilateral negotiation between agents *a* and *b* is considered using the evaluation functions for the responsive and deliberative mechanisms. Mechanisms for evaluating received state transition and deciding on which state transition to send

from the protocol are needed. For example on receiving a *request* with the set of issues $x^{t}_{b \to a}$, does agent *a* reply with an *offer*, *propose*, *suggest* or *reject* with $x^{t'}_{a \to b}$. The following sections describe our algorithms for combining the responsive and deliberative offer-generation mechanisms with the bilateral protocol to evaluate a received message and generate a message from the protocol accompanied with a set of issues.

8.4 Responsive Decision Making in a Bilateral Negotiation

In this section, a bilateral negotiation between agents *a* and *b* is considered, using the evaluation functions for the responsive mechanism. Let $x^{t}_{a \to b}$ be a vector of values from agent *a* to *b* at time *t* and $x^{t}_{a \to b}[j]$ be the value of issue *j* proposed by *a* to *b* at time *t*. Both agents start the negotiation at t_{start}, and they start counting from ($t_{start} = 0$). There is a local time for each negotiation process that starts with the first message being sent. t^{a}_{max} denotes the deadline for agent *a*.

An agent, *a*, evaluates a received set of issues and generates the set of issues it aims to respond with using its scoring function V^{a}. Agent *a* receives the set of issues $x^{t}_{b \to a}$ from agent *b* at time *t* and generates a response with the set $x^{t'}_{a \to b}$ to send to *b* at time *t'* where $t < t'$. $x^{t'}_{a \to b}$ is generated by concessions using a weighted mixture of the responsive mechanisms as described earlier. An algorithm for agent *a* sending a state-transition and $x^{t'}_{a \to b}$ to trigger the next state of the negotiation is given in the next section below.

8.4.1 Algorithm for a state transition using responsive mechanism

Initialisation for Algorithm 8.1

no-time-left = $t^{a}_{max} - t' < 2$

better-than = $V^{a}(x^{t}_{b \to a}) \geq V^{a}(x^{t'}_{a \to b})$

inputs: *current-state* /*current state triggered by *b*

distance /* nearness to a deal

set-of-possible-next-actions /*can be derived from protocol

$x^{t}_{b \to a}$ /*proposal from *b* to *a* at *t*

t^{a}_{max} /* deadline for *a*

V^{a} /*evaluation function for *a*

t' /* current time

$\{D^{a}_{1}, ..., D^{a}_{n}\}$ /*acceptable domains for each issue e.g. (min_i, max_i)

output: *next-action*$_a$, $x^{t'}_{a \to b}$ /*response from *a* with next-action and issues set*/

begin

(1) **if** (*t'* > t^{a}_{max}) **then** /*deadline for agent *a* has passed

(2) *next-action*$_a$ = ***timeout*** ; *exit*;

(3) **if not** (V^{a}($x^{t}_{b \to a}$) *lies within* D^{a}) **then** /* not acceptable from *b* *

(4) *sub-procedure not-acceptable* Algorithm 8.2; *exit*;

(5) **else** /* acceptable message from *b*/

(6) **if** (*current-state* = = ***offered*(*b*)** ∧ ***¬proposed*(*b*)**) ∨ ((*no-time-left* ∨ *better-than*) ∧ ***a.agree*** ∈ *set-of-possible-next-actions*) **then**

/* *a* considers an agree if no choice, no time or good offer from agent *b*

(7) *next-action*$_a$ = ***a.agree*** *with* $x^{t}_{b \to a}$

(8) **else if** (*no-time-left* ∨ *better-than*) **then** /*move towards a commitment */

(9) *next-action*$_a$ = ***a.offer*** *with* $x^{t}_{b \to a}$

(10) **else if** (*distance* = = *close* ∨ *middle*) **then** /*towards an offer, have time */

(11) **if** ***a.propose*** ∈ *set-of-possible-next-actions* **then**

(12) *next-action*$_a$ = ***a.propose*** *with* $x^{t'}_{a \to b}$

(13) **else** *next-action*$_a$ = ***a.counter-propose*** *with* $x^{t'}_{a \to b}$

(14) **else if** (*distance* = = *far*) **then** /*have time and wants more */

(15) **if** ***a.suggest*** ∈ *set-of-possible-next-actions* **then**

(16) *next-action*$_a$ = ***a.suggest*** *with* $x^{t'}_{a \to b}$

(17) **else** *next-action*$_a$ = ***a.request*** *with* $x^{t'}_{a \to b}$

(18) **endif**

(19) **endif**

end

Algorithm 8.1 Bilateral protocol with responsive mechanism

Inputs

A respondent agent, *a*, is aware of the *current-state* triggered by its opponent, *b*, with the proposed set of issues, $x^t_{b \to a}$ from agent *b* to agent *a* at time *t*. Agent *a* also knows its own reservation values, $\{D^a_1, \ldots, D^a_n\}$, for each issue. A quantitative issue, *i*, ranges from (min_i, max_i). From the bilateral protocol, an agent can derive the set of next possible actions. An agent knows its deadline, its evaluation function and the current time.

A set of issues from agent *b* can be compared with agent *a*'s goals to analyse the difference between what agent *a* has been proposed and what agent *a* will propose next. The variable *distance* may take three values: *close*, *middle* or *far* and is determined by the difference between $V^a(x^t_{b \to a})$ and $V^a(x^{t'}_{a \to b})$. $V^a(x^t_{b \to a})$ is the valuation of agent *a* of what agent *b* sent it and $V^a(x^{t'}_{a \to b})$ is the valuation of the response *a* will send to *b* according to agent *a*'s strategies. The variable *distance* measures with respect to the three intervals how far agent *a* and *b* are to an agreement.

$$close \quad \leftrightarrow \quad 0 < | V^a(x^{t'}_{a \to b}) - V^a(x^t_{b \to a}) | \leq 0.2$$

$$middle \quad \leftrightarrow \quad 0.2 < | V^a(x^{t'}_{a \to b}) - V^a(x^t_{b \to a}) | \leq 0.5$$

$$far \quad \leftrightarrow \quad 0.5 < | V^a(x^{t'}_{a \to b}) - V^a(x^t_{b \to a}) | \leq 1$$

Outputs

Given the current state triggered by agent *b* and the bilateral protocol, agent *a* will output the next action, $next\text{-}action_a$, for the next state transition. It will also send agent *b* a possibly new set of issues, $x^{t'}_{a \to b}$.

An agent, *a*, first checks that its deadline has not passed. If its deadline has expired, it sends a *timeout* message to close the negotiation. Otherwise, if the message from its opponent, *b*, is not acceptable then agent *a* follows Algorithm 8.2, where it generates a response with new values for the issues. An acceptable set of issues means that each issue in the set lies within the reservation values or in the qualitative set of instances that agent *a* has.

If the current state is *offered*(*b*) and not *proposed*(*b*), then agent *a* can only *agree* or *reject*. If the set of issues in agent *b*'s message is acceptable, then agent *a* agrees. If the set of issues from *b* is better than what agent *a* would have sent or if *a*'s deadline is close, then agent *a* sends an *agree*, if *agree* is possible. An agent does not risk bargaining when it does not have much time left to avoid an unsuccessful negotiation. The two conditions *better-than* and *no-time-left* are declared at the initialisation part.

If agent *a* cannot agree as its next action and if it has been sent a better set of issues than what it would have responded with or if the deadline is close (i.e. *no-time-left* or *better-than* are true), then agent *a* offers *b* with the set of issues that *b* previously sent.

If the above conditions are not met implying that the deadline is not close, then an agent may continue bargaining to attain a higher score. In that case, an agent decides on what state to trigger depending on the difference in what it received last from its opponent and the set of issues it aims to send next. This difference is given by *distance* and determines whether an agreement is *close*, *middle* or *far*. If they are *close* or *middle* to an agreement, then an agent moves to a higher level of commitment by sending *a.propose* or *a.counter-propose* with agent *a*'s new set of issues. If *distance* is *far*, several more steps can occur before termination and agent *a*'s next response is *a.request* or *a.suggest* with $x^{t'}_{a \to b}$.

8.4.2 Algorithm 8.1 in ANML

Theory 8.1 is a representation of Algorithm 8.1 in ANML. The theory is concise with respect to the algorithm. An agent sends a speech-act like message with a set

of issues e.g. Agent *x* sends an *offer* to agent *y* with the set of issues $x^t_{x \to y}$ at time *t*. We could represent such an offer action with the set of issues as a subscript or a parameter (that is $x.offer(x^t_{x \to y})$ or $x.offer_{x^t_{x \to y}}$). The resulting of offering $x^t_{x \to y}$ is the state $offered(x)_{x^t_{x \to y}}$. For the sake of conciseness we "factorise" the set of issues over the action *A* by agent *X* with the set of issues *S* leading to the state *B(X)* with the set of issues *S*. Thus, the following rule from the source state s_0

$s_0 \leftrightarrow [X.A_S]\ B_S(X)$ is abbreviated to $s_0 \leftrightarrow [X.A]\ B(X) \wedge S$.

For example with an offer and a set of issues $x^t_{x \to y}$, the rule

$requested(Y) \leftrightarrow [x.offer_{x^t_{x \to y}}]\ offered(X)_{x^t_{x \to y}}$ *is simplified to*

$$requested(Y) \leftrightarrow [X.offer]offered(X) \wedge x^t_{x \to y}$$

The ANML theory for Algorithm 8.1 is as below:

$$acceptable \leftrightarrow (\forall j{:}\, x^t_{b \to a})\ (min_j^a \le V^a(j) \le max_j^a \vee j \in D_a)$$

$$no\text{-}time\text{-}left \leftrightarrow (t^a_{max} - t' < 2)$$

$$better\text{-}than \leftrightarrow (V^a(x^t_{b \to a}) \ge V^a(x^{t'}_{a \to b}))$$

$$(open \wedge (t' > t^a_{max})) \leftrightarrow [timeout]\ timedout$$

$$(acceptable \wedge offered(b) \wedge \neg proposed(b)) \leftrightarrow ([a.agree]\ agreed) \wedge x^t_{b \to a}$$

$$(acceptable \wedge (no\text{-}time\text{-}left \vee better\text{-}than)) \leftrightarrow ([offered(b)?;\ a.agree]agreed \vee$$
$$[requested(b)?;\ a.offer\ ]\ offered(a))\ \wedge x^t_{b \to a}$$

$(acceptable \wedge (close \vee middle)) \leftrightarrow ($ $[(requested(b)?; a.propose) \cup$

$(proposed(b)?; a.counter\text{-}propose)\ ]\ proposed(a)\ ) \wedge\ x^{t'}_{a \to b}$

$(acceptable \wedge far) \leftrightarrow (\ [(requested(b)?; a.suggest) \cup (proposed(b)?;$

$a.request)]\ requested(a)) \wedge x^{t'}_{a \to b}$

Theory 8.1 Translating Algorithm 8.1 into an ANML theory

8.4.3 Algorithm when not-acceptable

Algorithm 8.2 is followed when the message from *b* to *a*, $x^{t}_{b \to a}$, is not acceptable to agent *a*. The initialisation part resembles Algorithm 8.1 for *no-time-left* and -*better-than*.

sub-procedure not-acceptable

inputs: *current-state* /*current state triggered by *b* */

distance, no-time-left /* time left to *a* and nearness to a deal */

set-of-possible-next-actions /*can be derived from bilateral protocol */

$x^{t}_{b \to a}$ /*proposal from *b* to *a* at *t* */

t^{a}_{max} /* deadline for *a* */

V^{a} /*evaluation function for *a* */

t' /* current time */

$\{D^{a}_{1}, ..., D^{a}_{n}\}$ /*acceptable issues domains e.g. (min_i, max_i)

output: *next-action*$_a$, $x^{t'}_{a \to b}$ /*response from *a* with next-action and issues set */

begin

(1) **if** (*current-state* = = ***offered*(*b*)** $\wedge$ **¬*proposed*(*b*)**) **then**

(2) *next-action*$_a$ = ***a.reject*** *with* $x^{t}_{b \to a}$; *exit*;

(3) **else if** *no-time-left* **then**

(4) **if** ***a.offer*** $\in$ *set-of-possible-next-actions* **then**

(5) *next-action*$_a$ = ***a.offer*** *with* $x^{t'}_{a \to b}$

(6) **else** *next-action*$_a$ = ***a.counter-offer*** *with* $x^{t'}_{a \to b}$

(7) **else if** *distance* == *close* **then**

(8) **if** ***a.propose*** ∈ *set-of-possible-next-actions* **then**

(9) *next-action*$_a$ = ***a.propose*** *with* $x^{t'}_{a \to b}$

(10) **else** *next-action*$_a$ = ***a.counter-propose*** *with* $x^{t'}_{a \to b}$

(11) **else if** *distance* == *middle* ∨ *far* **then**

(12) **if** ***a.suggest*** ∈ *set-of-possible-next-actions* **then**

(13) *next-action*$_a$ = ***a.suggest*** *with* $x^{t'}_{a \to b}$

(14) **else** *next-action*$_a$ = ***a.request*** *with* $x^{t'}_{a \to b}$

(15) **end**

end

Algorithm 8.2 Decisions by *a* when not-acceptable issues from *b*

The above algorithm shows the decisions made by agent *a* when it receives a non-acceptable set of issues and their values from *b*. Algorithm 8.1 calls Algorithm 8.2. Both algorithms have similar inputs and outputs. In Algorithm 8.2, agent *a* does not find it worthwhile to agree to $x^{t}_{b \to a}$ from *b*. Agent *a* responds with a more favourable set of issues, $x^{t'}_{a \to b}$. If the current state is *offered*(*b*) ∧ ¬*proposed*(*b*), *a* can only accept or reject and since $x^{t}_{b \to a}$ is not acceptable, then *a* rejects *b*'s set of issues. If there is no time left for further bargaining then agent *a* makes an *offer* or a *counter-offer* with $x^{t'}_{a \to b}$ as a penultimate step to termination. Otherwise if there is time left and if *distance* to an agreement is *close* then agent *a* triggers the *proposed* state with a proposal or counter-proposal. If *distance* is far or middle and there is time left for bargaining, then agent *a* triggers the *requested* state for further bargaining. Agent *a* always respond with $x^{t'}_{a \to b}$ since it can only accept what is inside its domain. Algorithm 8.2 is translated to an ANML theory for the strategy when an unacceptable message from agent *b* and for responding with $x^{t'}_{a \to b}$.

$$acceptable \leftrightarrow (\forall j: x^{t}_{b \to a})\ ((min_j^a \le V^a(j) \le max_j^a) \vee (j \in D_a))$$

$no\text{-}time\text{-}left \leftrightarrow (t^{a}_{max} - t' < 2)$

$(\neg acceptable \wedge offered(b) \wedge \neg proposed(b)) \leftrightarrow ([a.reject]rejected) \wedge x^{t}_{b \to a}$

$(\neg acceptable \wedge no\text{-}time\text{-}left) \leftrightarrow$ **(** $[(requested(b)?; a.offer) \cup (proposed(b)?; a.counter\text{-}offer)] offered(a)$**)** $\wedge x^{t'}_{a \to b}$

$(\neg acceptable \wedge close) \leftrightarrow ([(requested(b)?; a.propose) \cup (proposed(b)?; a.counter\text{-}propose)] proposed(a)) \wedge x^{t'}_{a \to b}$

$(\neg acceptable \wedge (far \vee middle)) \leftrightarrow ([(requested(b)?; a.suggest) \cup (proposed(b)?; a.request)] requested(a)) \wedge x^{t'}_{a \to b}$

Theory 8.2 Algorithm 8.2 as an ANML theory, unacceptable message

8.4.4 Generating Offers Using Responsive Mechanisms

A tactic is a function for determining the values of each issue in a set of issues according to a criterion such as time, resources, etc. Time and behaviour dependent tactics are used to generate $x^{t'}_{a \to b}$. Functions in section 8.2.2 are used for time-dependent and behaviour-dependent as responsive tactics.

An agent can generate its response by using a weighted combination of different tactics according to different criteria. An agent's strategy depends on which combination of tactics is used at an instant of a negotiation. An agent is allowed to change its rating about the importance of a tactic over time i.e. changing the weights associated to its tactics as a negotiation progresses. Four boolean variables, whose values depend on the amount of time left to the deadline, are defined: *much-time-left*, *middle-time-left*, *less-time-left* and *no-time-left*. The weight associated to a tactic is determined by the amount of time left and thus by the truth-value of the boolean variables. When an agent generates values for a set of issues using responsive tactics, it determines the time left until the deadline and which boolean variable is true. Then it uses the corresponding weights for each tactic at that time condition to compute the values in its set of issues. Let $t_{start} = 0$

and t_{max} be an agent's deadline. The following 4 equivalencies define *much-time-left*, *middle-time-left*, *less-time-left* and *no-time-left* and the weights of each tactic.

$$0 \le t < \frac{t_{\max}}{3} \quad \leftrightarrow \quad much\_time\_left \quad \leftrightarrow \quad \begin{cases} w_{time} = 0.2 \\ w_{behaviour} = 0.8 \end{cases}$$

$$\frac{t_{\max}}{3} \le t < \frac{2t_{\max}}{3} \quad \leftrightarrow \quad middle\_time\_left \quad \leftrightarrow \quad \begin{cases} w_{time} = 0.4 \\ w_{behaviour} = 0.6 \end{cases}$$

$$\frac{2t_{\max}}{3} \le t < t_{max} \text{-}1 \quad \leftrightarrow \quad less\_time\_left \quad \leftrightarrow \quad \begin{cases} w_{time} = 0.8 \\ w_{behaviour} = 0.2 \end{cases}$$

$$t_{max} \text{-}1 \le t \le t_{max} \quad \leftrightarrow \quad no\_time\_left \quad \leftrightarrow \quad \begin{cases} w_{time} = 1 \\ w_{behaviour} = 0 \end{cases}$$

For example, when $t_{max} = 60$ and $t = 34$ (in seconds), then *middle-time-left* is *true* and the weight associated to each tactic is $w_{time} = 0.4$ and $w_{behaviour} = 0.6$. If the time-dependent tactic gives $(x_{a \to b})^t$ [*price*] = 32 and the behaviour-dependent tactic gives $(x_{a \to b})^t$ [*price*] = 40, then the response generated by agent *a* to agent *b* is $(0.4 \times 32) + (0.6 \times 40) = 36.8$

The weights for each tactic are modeled such than when an agent has time left, it prefers an imitative behaviour to a time-dependent tactic. But as its deadline approaches, it increasingly favours a time-dependent tactic and is more concerned to concede closer to its deadline. A responsive strategy is computationally and resource inexpensive and shows reasonable performance.

8.5 Deliberative Decision Making in a Bilateral Negotiation

The responsive mechanism involves concessions over time and thus fails to find joint gains and Pareto optimal solutions. Solutions for joint gains are beneficial to all participants. Changing a Pareto solution would involve at least an agent gaining less. The responsive mechanism cannot discriminate between sets of issues with possibly different values for an issue but with the same overall value for the set. The deliberative mechanisms described in [Faratin and al. 1999] and in section 8.2.3 use *trade-offs* to achieve more win-win results. An agent, *a*, making a trade-off seeks a set of issues which has the same value to itself but the

set is possibly more beneficial to its opponent than *a*'s previous proposal. This is achieved by lowering the values on some issues while increasing others.

The strategy for an agent using the bilateral protocol with the deliberative mechanisms in section 8.2.3 is given below. The algorithms describe an agent's decision making on the next state transition from the bilateral protocol. An agent may concede if the deadline is close or its opponent has ceased conceding.

8.5.1 A bilateral Negotiation using Deliberative mechanisms

Agent, *a*, previously proposed $x_{a\to b}^{t_{n-1}}$ to agent *b* at time t_{n-1} and received $x_{b\to a}^{t_n}$ from agent *b* at time t_n. At time t_{n+1}, agent *a* will respond with $x_{a\to b}^{t_{n+1}}$ and an action from the protocol to trigger the next state transition, according to Algorithm 8.3.

better-than $\leftrightarrow V^a(x_{b\to a}^{t_n}) \geq V^a(x_{a\to b}^{t_{n-1}})$

Algorithm 8.3

inputs: *current-state* /*current state triggered by *b**/

distance /* nearness to a deal*/

set-of-possible-next-actions /*can be derived from protocol*/

$x_{a\to b}^{t_{n-1}}$, $x_{b\to a}^{t_n}$ /*previous set of issues exchanged*/

t_{max}^a /* deadline for *a* */

V^a /*evaluation function for *a**/

t_n /* current time*/

$D^a = \{D_1^a, ..., D_n^a\}$ /*acceptable domains for each issue

output: *next-action*$_a$, $x_{a\to b}^{t_{n+1}}$ /**a* responds with next-action and issues set*/

begin

(1) **if** ($t_{n+1} > t_{max}^a$) **then** /*deadline has passed */

(2) *next-action*$_a$ = ***timeout*** ; *exit*;

(3) **if not** ($V^a(x_{b\to a}^{t_n})$ *lies within* D^a) **then** /* $x_{b\to a}^{t_n}$ is not acceptable to *a**/

(4) *sub-procedure not-acceptable-deliberative* Algorithm 8.7; *exit*; **(I)**

(5) **else** /* acceptable message from *b**/

(6) **if** (*current-state* = = ***offered*(*b*)** ∧ **¬*proposed*(*b*)**) ∨ ((*no-time-left* ∨ *better-than*) ∧ ***a.agree*** ∈ *set-of-possible-next-actions*) **then**

(7) *next-action*$_a$ = ***a.agree*** *with* $x_{b\to a}^{t_n}$

(8) **else if** (*no-time-left* ∨ *better-than*) **then**

(9) *next-action*$_a$ = ***a.offer*** *with* $x_{b\to a}^{t_n}$

(10) **else if** (*distance* = = *far*) **then**

(11) *sub-procedure acceptable-and-far-distance* Algorithm 8.6 **(II)**

(12) **else if** (*distance* = = *middle*) **then**

(13) *sub-procedure acceptable-and-middle-distance* Algorithm 8.5 **(III)**

(14) **else if** (*distance* = = *close*) **then**

(15) *sub-procedure acceptable-and-close-distance* Algorithm 8.4 **(IV)**

(16) **endif**

(17) **endif**

end

Algorithm 8.3 Bilateral protocol with deliberative mechanism

Agent *a* first checks whether its deadline is past and if so sends a timeout. Otherwise if the message from agent *b* is acceptable and there is no time left or *b*'s set of issues is better than or equal to *a*'s previous proposal, then agent *a* either agrees with *b* on $x_{b\to a}^{t_n}$ or *a* sends an offer to *b* with $x_{b\to a}^{t_n}$. If agent *b* made an acceptable *offer* to *a*, then agent *a* accepts *b*'s offer. If $x_{b\to a}^{t_n}$ is not acceptable to agent *a* then the Algorithm 8.7 is followed. If $x_{b\to a}^{t_n}$ is acceptable to agent *a* and agent *a* is not pressed to agree, then agent *a* decides on its next response according to its previous response at t_{n-1} and what *b* sent to *a* at t_n. The variable *distance* can take values *far, close* or *middle* to define the difference between the valuations of agent *a* for $x_{a\to b}^{t_{n-1}}$ and $x_{b\to a}^{t_n}$. *distance* is modeled in terms of the difference between the scores of the last set of issues received from agent *b* and what agent *a*'s previously sent.

$$close \quad \leftrightarrow \quad 0 < |\, V^a(x_{a\to b}^{t_{n-1}}) - V^a(x_{b\to a}^{t_n}) \,| \le 0.2$$

$$middle \quad \leftrightarrow \quad 0.2 < |\, V^a(x_{a\to b}^{t_{n-1}}) - V^a(x_{b\to a}^{t_n}) \,| \le 0.5$$

$$far \quad \leftrightarrow \quad 0.5 < |\, V^a(x_{a\to b}^{t_{n-1}}) - V^a(x_{b\to a}^{t_n}) \,| \le 1$$

Algorithm 8.3 can be translated into a theory in ANML for agent *a* to verify and reason about.

$$acceptable \leftrightarrow (\forall j\text{:}\ x^{t_n}_{b\to a})\ (\ (min_j^a \le V^a(j) \le max_j^a) \vee (j \in D_a)\ )$$

$$(open \wedge (t_{n+1} > t^a_{max})) \leftrightarrow [timeout]\ timedout$$

$$(acceptable \wedge offered(b) \wedge \neg proposed(b)) \leftrightarrow ([a.agree]\ agreed) \wedge x^{t_n}_{b\to a}$$

$$(acceptable \wedge (no\text{-}time\text{-}left \vee better\text{-}than)) \leftrightarrow (\ ([offered(b)?;\ a.agree]agreed) \vee\ ([requested(b)?;\ a.offer]\ offered(a))\ )\ \wedge x^{t_n}_{b\to a}$$

$$(acceptable \wedge close) \leftrightarrow [\{a, b\}.\ acceptable\text{-}and\text{-}close\text{-}distance\text{-}process]sent$$

$$(acceptable \wedge middle) \leftrightarrow [\{a, b\}.\ acceptable\text{-}and\text{-}middle\text{-}distance\text{-}process]sent$$

$$(acceptable \wedge far) \leftrightarrow [\{a, b\}.\ acceptable\text{-}and\text{-}far\text{-}distance\text{-}process]sent$$

Theory 8.3 Bilateral protocol with deliberative strategy

In contrast to the responsive strategy, the deliberative mechanism is time and computationally more costly. The time for generating the values for a set of issues must be taken into account. Performance depends on the size of the space of potential agreements and on the hardware computational facilities. An agent should avoid missing its deadline while generating an offer. For example, if agent *a* takes on average 4 seconds to generate an offer and there is 4 seconds before its deadline, then it must settle for a quick decision with an *offer* or an *agree*. It also adopts a less time-costly strategy such as the responsive mechanism.

During a negotiation, an agent, using the deliberative strategy, notes the maximum time, t_{gen}, it has needed to generate a proposal so far. An agent has 5 Boolean variables to deal with the amount of time left before an agent's deadline, *out-of-time*, *no-time-left*, *less-time-left*, *middle-time-left* and *much-time-left*. These variables are defined in terms of t_{gen} and the time left to an agent's deadline t_{max}. Let the current time be *t*.

$$t_{max} < t \quad \leftrightarrow \quad out\text{-}of\text{-}time$$

$$t_{max} - t_{gen} \leq t \leq t_{max} \quad \leftrightarrow \quad no\_time\_left$$

$$\frac{2(t_{\max} - t_{gen})}{3} \leq t < t_{max} - t_{gen} \quad \leftrightarrow \quad less\_time\_left$$

$$\frac{t_{\max} - t_{gen}}{3} \leq t < \frac{2(t_{\max} - t_{gen})}{3} \quad \leftrightarrow \quad middle\_time\_left$$

$$0 \leq t < \frac{t_{\max} - t_{gen}}{3} \quad \leftrightarrow \quad much\_time\_left$$

A function *deadlock*(*R*) checks the similarity between the last *R* proposals from an opponent. *deadlock*(*R*) returns *true* if an opponent has not been conceding in its last *R-1* proposals. The opponent has been proposing set of issues of almost the same overall score to agent *a*.

The deliberative mechanism will make an agent bargain until it has reached an agreement or it is close to its deadline. An agent triggers an *offered* state if it does not have enough time to bargain. An agent sends a proposal instead of an offer if it has enough time. An agent *a* decides if it has enough time to send a proposal at t_{n+1} by estimating the time when it will need to respond next i.e. t_{n+3}. Agent *a* responds at t_{n+1}, agent *b* at t_{n+2} and agent *a* next responds at t_{n+3}. Agent *a* has to calculate whether t_{n+3} is still less than t_{max}. Time t_{n+3} is dependent on t_{n+2} i.e. on how long it takes for its opponent to respond. Therefore, agent *a* notes the maximum time its opponent takes to generate a proposal and respond, i.e. t_{opp_gen}. The time t_{opp_gen} is dynamic and varies according to the opponent's computational resources and conditions. An agent calculates t_{n+3} as (t_{n+1} + t_{opp_gen}) where t_{n+1} is the time agent *a* responds.

$$t_{max} - t_{gen} \leq t_{n+3} \quad \leftrightarrow \quad \neg\ time\text{-}for\text{-}proposal$$

$$t_{n+3} < t_{max} - t_{gen} \quad \leftrightarrow \quad time\text{-}for\text{-}proposal$$

The condition *time-for-proposal* is *false* if an agent computes that t_{n+3} will be outside or too close to its deadline and *true* otherwise. If *time-for-proposal* is *true*, at the current time an agent does not make an offer since it can bargain for one more step.

8.5.2 Generating values with a deliberative mechanism

An agent, *a*, using the deliberative mechanism, can either concede or trade-off in generating their response $x_{a\to b}^{t_{n+1}}$. The algorithms below prescribe the conditions when an agent concedes or chooses to trade-off. An agent, *a*, concedes by generating a contract $x_{a\to b}^{t_{n+1}}$ with a lower score than its previous one, $x_{a\to b}^{t_{n-1}}$. An agent *a* decides to trade-off by generating a contract $x_{a\to b}^{t_{n+1}}$ that has the same score as its previous one $x_{a\to b}^{t_{n-1}}$, but which can be more worthwhile to its opponent. Section 8.2.3 gives various trade-off mechanisms.

Concessions

An agent choosing to concede has to decide on how much to concede and what set of issues and values to propose next. The extent an agent concedes depends (is inversely proportional) on the amount of time left until its deadline. The more time an agent has the less it concedes, and only so as to trigger its opponent to concede and to move from a deadlock. On the other hand if *less-time-left* is true, then an agent should concede enough to get its offer accepted, but no more or less than its reservation values. An agent must not propose a contract that has a higher score for its opponent than its opponent's own previous proposal. Thus, an agent concedes depending on the amount of time left and its opponent's last proposal. The difference between an agent's last proposal and its opponent's response is used to calculate an agent's concession as given in the function below:

$$V^a(x_{a\to b}^{t_{n+1}}) = V^a(x_{a\to b}^{t_{n-1}}) - Dif \times \frac{\min(t, t_{\max})}{t_{\max}}$$

where $Dif = V^a(x_{a\to b}^{t_{n-1}}) - V^a(x_{b\to a}^{t_n})$

where t is the current time and t_{max} is the deadline of agent *a*.

8.5.3 Algorithm for *acceptable* and *close*

This algorithm prescribes the decision making of agent *a* when agent *b* has sent an acceptable set of issues and the *distance* between agent *a*'s previous proposal and

b's latest proposal is close. This algorithm is called at point (IV) in Algorithm 8.3.

inputs: *current-state* /*current state triggered by *b**/
distance /* nearness to a deal*/
set-of-possible-next-actions /*can be derived from protocol*/
$x_{a\to b}^{t_{n-1}}$, $x_{b\to a}^{t_n}$ /* previous set of issues exchanged */
t_{max}^{a} /* deadline for *a* */
V^a /*evaluation function for *a**/
t_n /* current time*/
$D^a = \{D_1^a, ..., D_n^a\}$ /*acceptable domains for each issue
output: *next-action*$_a$, $x_{a\to b}^{t_{n+1}}$ /*response from *a* at t_{n+1}

begin
(1) **if** (*less-time-left* ∧ *deadlock*(*2*)) ∨ (*much-time-left* ∧ *deadlock*(*4*)) ∨ (*middle-time-left* ∧ *deadlock*(*3*)) **then**
(2) *a* **concedes** in generating $x_{a\to b}^{t_{n+1}}$
(3) **else**
(4) *a* **trade-offs** in generating $x_{a\to b}^{t_{n+1}}$
(5) **endif**
(6) **if** *time-for-proposal* **then**
(7) **if** ***a.propose*** ∈ *set-of-possible-next-actions* **then**
(8) *next-action*$_a$ = ***a.propose*** *with* $x_{a\to b}^{t_{n+1}}$
(9) **else** *next-action*$_a$ = ***a.counter-propose*** *with* $x_{a\to b}^{t_{n+1}}$
(10) **else** /* t_{n+3} is close to the deadline */
(11) **if** ***a.offer*** ∈ *set-of-possible-next-actions* **then**
(13) *next-action*$_a$ = ***a.offer*** *with* $x_{a\to b}^{t_{n+1}}$
(14) **else** *next-action*$_a$ = ***a.counter-offer*** *with* $x_{a\to b}^{t_{n+1}}$
(15) **endif**
end

Algorithm 8.4 Decisions when *acceptable* and *distance* is *close*

Algorithm 8.4 shows agent *a*'s decisions when the set of issues from agent *b* is acceptable and both agents are *close* to an agreement. Using a deliberative

strategy, agent *a* bargains until its deadline is close. Agent *a* decides on its response depending on the amount of time left and whether there is deadlock because agent *b* has not conceded in the previous steps. Agent *a* checks for a *deadlock* but is more strict in conceding depending on the amount of time left. Agent *a* concedes when generating $x_{a\to b}^{t_{n+1}}$ if there is *less-time-left* and agent *b* has not conceded in its last offer. In *deadlock*(*2*) the last two set of issues from agent *b* are compared. Agent *a* also concedes if *much-time-left* and *deadlock*(4) is *true* or *middle-time-left* or *deadlock*(3) is true. Otherwise, agent *a* uses the trade-off mechanism.

Having decided whether to concede or not for $x_{a\to b}^{t_{n+1}}$, agent *a* next chooses its action that will trigger the next state of the negotiation. Since both agents are close to an agreement, then the next state should allow for a respondent to agree. If agent *a* has enough time for a proposal i.e. *time-for-proposal* is *true*, then agent *a* either makes a proposal or counter-proposal with $x_{a\to b}^{t_{n+1}}$. This will allow agent *b* to agree or to make another response where agent *a* can then agree or offer. If *time-for-proposal* is *false*, then agent *a* has no more time for bargaining and has to trigger the *offered* state with an offer or counter-offer with $x_{a\to b}^{t_{n+1}}$. Algorithm 8.4 is translated in ANML for the *acceptable-and-close-distance-process* process.

((*less-time-left* ∧ *deadlock*(2)) ∨ (*much-time-left* ∧ *deadlock*(*4*)) ∨ (*middle-time-left* ∧ *deadlock*(*3*)))**)** ↔ [*a.concedes-generating-* $x_{a\to b}^{t_{n+1}}$] *generated*(*a*)

¬ **(**(*less-time-left* ∧ *deadlock*(2)) ∨ (*much-time-left* ∧ *deadlock*(*4*)) ∨ (*middle-time-left* ∧ *deadlock*(*3*)))**)** ↔ [*a.trade-offs-generating-* $x_{a\to b}^{t_{n+1}}$] *generated*(*a*)

(*acceptable* ∧ *close* ∧ *time-for-proposal*) ↔ [(*requested*(*b*)?; *a.propose*) ∪ (*proposed*(*b*)?; *a.counter-propose*)] *proposed*(*a*)

(*acceptable* ∧ *close* ∧ ¬*time-for-proposal*) ↔ [(*requested*(*b*)?; *a.offer*) ∪ (*proposed*(*b*)?; *a.counter-offer*)] *offered* (*a*)

Theory 8.4 Decisions for *acceptable-and-close-distance-process*

8.5.4 Algorithm for *acceptable* and *middle*

The following algorithm shows the decision process for agent a after an acceptable set of issues from b and the difference between b's set of issues and that in a's message is *middle*. It is called at point (III) in Algorithm 8.3.

inputs: *current-state* /*current state triggered by *b**/
distance /* nearness to a deal*/
set-of-possible-next-actions /*can be derived from protocol*/
$x_{a\to b}^{t_{n-1}}$, $x_{b\to a}^{t_n}$ /* previous proposals*/
t_{max}^{a} /* deadline for *a* */
V^a /*evaluation function for *a**/
t_n /* current time*/
$D^a = \{D_1^a, \ldots, D_n^a\}$ /*acceptable domains for each issue */

output: *next-action*$_a$, $x_{a\to b}^{t_{n+1}}$ /*response from *a* */

begin
(1) **if** *less-time-left* ∨ (*much-time-left* ∧ *deadlock*(*3*)) ∨ (*middle-time-left* ∧ *deadlock*(*2*)) **then**
(2) *a* **concedes** in generating $x_{a\to b}^{t_{n+1}}$
(3) **else** *a* **trade-offs** in generating $x_{a\to b}^{t_{n+1}}$
(4) **endif**
(5) **if** ¬*time-for-proposal* **then**
(6) **if** ***a.offer*** ∈ *set-of-possible-next-actions* **then**
(7) *next-action*$_a$ = ***a.offer*** *with* $x_{a\to b}^{t_{n+1}}$
(8) **else** *next-action*$_a$ = ***a.counter-offer*** *with* $x_{a\to b}^{t_{n+1}}$
(9) **else** /**time-for-proposal* is *true**/
(10) **if** *much-time-left* **then** /*can bargain*/
(11) **if** ***a.suggest*** ∈ *set-of-possible-next-actions* **then**
(12) *next-action*$_a$ = ***a.suggest*** *with* $x_{a\to b}^{t_{n+1}}$
(13) **else** *next-action*$_a$ = ***a.request*** *with* $x_{a\to b}^{t_{n+1}}$
(14) **else** /*not much time left*/
(16) **if** ***a.propose*** ∈ *set-of-possible-next-actions* **then**
(17) *next-action*$_a$ = ***a.propose*** *with* $x_{a\to b}^{t_{n+1}}$

(18) **else** *next-action*$_a$ = ***a.counter-propose*** *with* $x_{a\to b}^{t_{n+1}}$

(19) **endif**

(20) **endif**

end

Algorithm 8.5 Decisions when *acceptable* and *distance* is *middle*

Algorithm 8.5 is used when agent *b* sends an acceptable message to agent *a* and the difference between *b*'s set of issues and that in agent *a*'s last message is *middle* (as defined by *distance*). Agent *a* first checks the amount of time left in order to decide whether to concede or to trade-off in generating $x_{a\to b}^{t_{n+1}}$. If *less-time-left* is *true* i.e. a deadline is close then agent *a* concedes to favour an agreement soon. If *middle-time-left*, then agent *a* concedes if agent *b* has not conceded in its last proposal (i.e. *deadlock*(2)), otherwise *a* tradeoffs. Agent *a* is more strict in conceding if it has *much-time-left* and therefore concedes only if *deadlock*(3), and tradeoffs otherwise.

In choosing on what action to trigger the next current state, agent *a* makes an offer or a counter-offer if it does not have any time for bargaining. If it has enough time for another proposal and *much-time-left*, then agent *a* bargains through *requests* or *suggests*. If not *much-time-left* but it still has time for a proposal, then agent *a* triggers the *proposed* state. Algorithm 8.5 is represented in ANML with the conditions of *acceptable* and *middle* for *acceptable-and-middle-distance-process* process.

$$(\textit{less-time-left} \vee (\textit{much-time-left} \wedge \textit{deadlock}(3)) \vee (\textit{middle-time-left} \wedge \textit{deadlock}(2)) \leftrightarrow [\textit{a.concedes-generating-}\, x_{a\to b}^{t_{n+1}}]\, \textit{generated}(a)$$

$$\neg(\textit{less-time-left} \vee (\textit{much-time-left} \wedge \textit{deadlock}(3)) \vee (\textit{middle-time-left} \wedge \textit{deadlock}(2)) \leftrightarrow [\textit{a.trade-offs-generating-}\, x_{a\to b}^{t_{n+1}}]\, \textit{generated}(a)$$

$$(\textit{time-for-proposal} \wedge \textit{much-time-left}) \leftrightarrow [(\textit{requested}(b)?;\, \textit{a.suggest}) \cup (\textit{proposed}(b)?;\, \textit{a.request})]\, \textit{requested}(a)$$

(*time-for-proposal* ∧ ¬*much-time-left*) ↔ [(*requested*(*b*)?; *a.propose*) ∪ (*proposed*(*b*)?; *a.counter-propose*)] *proposed*(*a*)

¬*time-for-proposal* ↔ [(*requested*(*b*)?; *a.offer*) ∪ (*proposed*(*b*)?; *a.counter-offer*)] *offered*(*a*)

Theory 8.5 Decisions when *acceptable* and *distance* is *middle*

8.5.5 Algorithm for *acceptable* and *far*

The last case of an acceptable proposal from agent *b* to *a* is when the *distance* is *far*. This algorithm is called at point (II) in Algorithm 8.3.

begin
(1) **if** *less-time-left* ∨ *middle-time-left* ∨ (*much-time-left* ∧ *deadlock*(*2*)) **then**
(2) *a* **concedes** in generating $x^{t_{n+1}}_{a\to b}$
(3) **else** *a* **trade-offs** in generating $x^{t_{n+1}}_{a\to b}$
(4) **endif**
(5) **if** ¬*time-for-proposal* **then**
(6) **if** ***a.offer*** ∈ *set-of-possible-next-actions* **then**
(7) *next-action*$_a$ = ***a.offer*** *with* $x^{t_{n+1}}_{a\to b}$
(8) **else** *next-action*$_a$ = ***a.counter-offer*** *with* $x^{t_{n+1}}_{a\to b}$
(9) **else** /**time-for-proposal* is *true**/
(10) **if** *middle-time-left* ∨ *much-time-left* **then** /*can bargain*/
(11) **if** ***a.suggest*** ∈ *set-of-possible-next-actions* **then**
(12) *next-action*$_a$ = ***a.suggest*** *with* $x^{t_{n+1}}_{a\to b}$
(13) **else** *next-action*$_a$ = ***a.request*** *with* $x^{t_{n+1}}_{a\to b}$
(14) **else** /*if less time left*/
(21) **if** ***a.propose*** ∈ *set-of-possible-next-actions* **then**
(22) *next-action*$_a$ = ***a.propose*** *with* $x^{t_{n+1}}_{a\to b}$
(23) **else** *next-action*$_a$ = ***a.counter-propose*** *with* $x^{t_{n+1}}_{a\to b}$
(24) **endif**
(25) **endif**
end

Algorithm 8.6 Decisions when *acceptable* and *distance* is *far*

Agent *a* concedes if there is *less-time-left* or *middle-time-left* or if there is much time left but its opponent has not conceded in its last set of issues. Since the distance towards an agreement is far, then agent *a* is more inclined to concede than in the previous algorithms. If there is no deadlock and there is much time left, then an agent uses the trade-off mechanism.

To decide on the next state transition, agent *a* checks if it has time to bargain and its deadline is not reached at t_{n+3}. If there is no time for a proposal, then agent *a* triggers the *offered* state as a take-it-or-leave it ultimatum. If *time-for-proposal* is *true* then agent *a* sends a proposal or a request depending on the time left to its deadline. At *much* or *middle time left*, an agent bargains by triggering a *requested* state. If *less-time-left* with still time for a proposal, then an agent triggers the *proposed* state. The ANML theory for Algorithm 8.6 is given below for *acceptable-and-far-distance-process* process.

(*less-time-left* ∨ *middle-time-left* ∨ (*much-time-left* ∧ *deadlock*(*2*))**)** ↔

[*a.concedes-generating-* $x_{a\to b}^{t_{n+1}}$] *generated*(*a*)

¬**(***less-time-left* ∨ *middle-time-left* ∨ (*much-time-left* ∧ *deadlock*(*2*))**)** ↔

[*a.trade-offs-generating-* $x_{a\to b}^{t_{n+1}}$] *generated*(*a*)

(*time-for-proposal* ∧ (*middle-time-left* ∨ *much-time-left*)) ↔ [(*requested*(*b*)?; *a.suggest*) ∪ (*proposed*(*b*)?; *a.request*)] *requested*(*a*)

(*time-for-proposal* ∧ ¬(*middle-time-left* ∨ *much-time-left*)) ↔

[(*requested*(*b*)?; *a.propose*) ∪ (*proposed*(*b*)?; *a.counter-propose*)] *proposed*(*a*)

¬*time-for-proposal* ↔ [(*requested*(*b*)?; *a.offer*) ∪ (*proposed*(*b*)?; *a.counter-offer*)] *offered*(*a*)

Theory 8.6 Decisions when *acceptable* and *distance* is *far*

8.5.6 Algorithm for a *not-acceptable* set of issues

A non-acceptable set of issues to an agent, *a*, implies that one or more values in that set is out of the range of what agent *a* can accept. Partial agreement to a negotiation is not considered (where an agent agrees to a subset of the set of issues and continues to negotiate on the rest). In the given algorithms, an agent cannot agree to a non-acceptable contract. An agent compromises when it receives non-acceptable contracts as it is unsure of whether there is an intersection between its and its opponent's goals. Algorithm 8.7 is used when agent, *a*, is using the deliberative mechanism and received a non-acceptable response from its opponent. It is called at point (I) in Algorithm 8.3 and itself calls other procedures depending on the distance towards an agreement. Algorithm 8.7 gives the decision making when an agreement is *far*.

begin

(1) **if** (*current-state* = = ***offered*(*b*)** ∧ **¬*proposed*(*b*)** ∨ *no-time-left*) **then**

(2) *next-action*$_a$ = ***a.reject*** $x^{t_n}_{b\to a}$; *exit*;

(3) **else if** (*distance* = = *middle*) **then**

(4) *sub-procedure non-acceptable-and-middle-distance* Algorithm 8.8 **(I)**

(5) **else if** (*distance* = = *close*) **then**

(6) *sub-procedure non-acceptable-and-close-distance* Algorithm 8.9 **(II)**

(7) **else if** (*distance* = =*far*) **then**

(8) *a* **concedes** in generating $x^{t_{n+1}}_{a\to b}$

(9) **if** *time-for-proposal* **then**

(10) **if** ***a.suggest*** ∈ *set-of-possible-next-actions* **then**

(11) *next-action*$_a$ = ***a.suggest*** *with* $x^{t_{n+1}}_{a\to b}$

(12) **else** *next-action*$_a$ = ***a.request*** *with* $x^{t_{n+1}}_{a\to b}$

(13) **else** /* t_{n+3} is close to the deadline */

(14) **if** ***a.offer*** ∈ *set-of-possible-next-actions* **then**

(16) *next-action*$_a$ = ***a.offer*** *with* $x^{t_{n+1}}_{a\to b}$

(17) **else** *next-action*$_a$ = ***a.counter-offer*** *with* $x^{t_{n+1}}_{a\to b}$

(18) **endif** /*time-for-proposal*/

(19) **endif**

end

Algorithm 8.7 Decisions when *not-acceptable* and when distance is *far*

If agent *b* has *offered* a non-acceptable set of issues or if there is no more time left, then an agent can only *reject b*'s set of issues. However if there is time left, then an agent decides on its response according to the time and to the distance towards an agreement (i.e. the difference between its and the opponents previous response).

Algorithm 8.7 gives agent *a*'s decision making when the *distance* is *far*. An agent concedes in this situation to be closer to an acceptable region for the values in a set of issues. The state transition by agent *a* depends on the amount of time left. If there is not enough time left for further messages after agent *b* responds at t_{n+2}, then agent *a* sends an *offer* or a *counter-offer* with $x_{a\to b}^{t_{n+1}}$. Otherwise if there is time left then agent *a* triggers the *requested* state since *distance* is *far* and substantial interaction is needed to reach an agreement. It is unlikely that agent *b* would agree to a *propose* from agent *a* since the distance towards an agreement is *far*. The corresponding ANML theory for Algorithm 8.7 is given below.

$(\neg acceptable \wedge (\ (offered(b) \wedge \neg proposed(b)) \vee no\text{-}time\text{-}left)) \leftrightarrow [a.reject]$ *rejected*

$(\neg acceptable \wedge close) \leftrightarrow [\{a, b\}.\ not\text{-}acceptable\text{-}and\text{-}close\text{-}distance\text{-}process]$ *sent*

$(\neg acceptable \wedge middle) \leftrightarrow [\{a, b\}.\ not\text{-}acceptable\text{-}and\text{-}middle\text{-}distance\text{-}process]$ *sent*

$(\neg acceptable \wedge far) \leftrightarrow [a.concedes\text{-}generating\text{-}x_{a\to b}^{t_{n+1}}]\ generated(a)$

$time\text{-}for\text{-}proposal \leftrightarrow [(requested(b)?;\ a.suggest) \cup (proposed(b)?;\ a.request)]$ *requested*(a)

$\neg time\text{-}for\text{-}proposal \leftrightarrow [(requested(b)?;\ a.offer) \cup (proposed(b)?;\ a.counter\text{-}offer)]\ offered(a)$

Theory 8.7 Decisions when *not-acceptable* and when distance is *far*

8.5.7 Algorithm for not-acceptable and middle

The algorithm in this section describes agent *a*'s decision making when *b* has sent it an unacceptable set of values and the difference between *a*'s and *b*'s last responses give a *middle distance*. Algorithm 8.8 is called at point (I) in Algorithm 8.7.

begin
(1) **if** *less-time-left* ∨ (*much-time-left* ∧ *deadlock*(*2*)) ∨ *middle-time-left* **then**
(2) *a* **concedes** in generating $x^{t_{n+1}}_{a\to b}$
(3) **else**
(4) *a* **trade-offs** in generating $x^{t_{n+1}}_{a\to b}$
(5) **endif**
(6) **if** ¬*time-for-proposal* **then**
(7) **if** ***a.offer*** ∈ *set-of-possible-next-actions* **then**
(8) *next-action*$_a$ = ***a.offer*** *with* $x^{t_{n+1}}_{a\to b}$
(9) **else** *next-action*$_a$ = ***a.counter-offer*** *with* $x^{t_{n+1}}_{a\to b}$
(10) **else** /**time-for-proposal* is *true***/*
(11) **if** *less-time-left* **then** /* commits to a proposal*/
(12) **if** ***a.propose*** ∈ *set-of-possible-next-actions* **then**
(13) *next-action*$_a$ = ***a.propose*** *with* $x^{t_{n+1}}_{a\to b}$
(14) **else** *next-action*$_a$ = ***a.counter-propose*** *with* $x^{t_{n+1}}_{a\to b}$
(15) **else** /*more time for bargaining*/
(16) **if** ***a.suggest*** ∈ *set-of-possible-next-actions* **then**
(17) *next-action*$_a$ = ***a.suggest*** *with* $x^{t_{n+1}}_{a\to b}$
(18) **else** *next-action*$_a$ = ***a.request*** *with* $x^{t_{n+1}}_{a\to b}$
(19) **endif**
(20) **endif**
end

Algorithm 8.8 Decisions when *not-acceptable* and *middle* distance

Agent *a* concedes if the deadline is near when *less-time-left* or *middle-time-left*, so as to approach an agreement. If agent *a* has *much-time-left* or agent *b* has conceded in its last message, then *a* uses the trade-off mechanism to generate its response.

In choosing a state transition, an agent checks if it needs to make an offer right away because of lack of time. If not, then it can, as for the other algorithms, either trigger a *requested* state or move towards a proposal depending on how close is its deadline. Theory 8.8 is the corresponding ANML theory for Algorithm 8.8 for the process *not-acceptable-and-middle-distance-process*.

(*less-time-left* ∨ (*much-time-left* ∧ *deadlock*(*2*)) ∨ *middle-time-left***)** ↔
[*a.concedes-generating-* $x_{a\to b}^{t_{n+1}}$] *generated*(*a*)

(*much-time-left* ∧ ¬*deadlock*(*2*)) ↔ [*a.trade-offs-generating-* $x_{a\to b}^{t_{n+1}}$]
generated(*a*)

(*time-for-proposal* ∧ (*much-time-left* ∨ *middle-time-left*)) ↔ [(*requested*(*b*)?;
a.suggest) ∪ (*proposed*(*b*)?; *a.request*)] *requested*(*a*)

(*time-for-proposal* ∧ *less-time-left*) ↔ [(*requested*(*b*)?; *a.propose*) ∪
(*proposed*(*b*)?; *a.counter-propose*)] *proposed*(*a*)

¬*time-for-proposal* ↔ [(*requested*(*b*)?; *a.offer*) ∪ (*proposed*(*b*)?; *a.counter-offer*)] *offered*(*a*)

Theory 8.8 Decisions when *not-acceptable* and *middle* distance

8.5.8 Algorithm for not-acceptable and close

Algorithm 8.9 is followed when agent *a* estimates that both agents are close to an agreement but agent *b* has sent a non-acceptable message to agent *a*. This is the case when for example two agents are proposing sets of issues of almost acceptable scores but there are some proposed values that are out of the range of acceptability of an agent. Algorithm 8.9 is called at point (II) in Algorithm 8.7.

begin
(1) **if** (*less-time-left* ∨ (*much-time-left* ∧ *deadlock*(*3*)) ∨ (*middle-time-left* ∧ *deadlock*(*2*)))
(2) *a* **concedes** in generating $x_{a\to b}^{t_{n+1}}$
(3) **else**

(4) *a* **trade-offs** in generating $x^{t_{n+1}}_{a\to b}$

(5) **endif**

(6) **if** *time-for-proposal* **then**

(7) **if** *much-time-left* **then**

(8) **if** ***a.suggest*** ∈ *set-of-possible-next-actions* **then**

(9) *next-action*$_a$ = ***a.suggest*** *with* $x^{t_{n+1}}_{a\to b}$

(10) **else** *next-action*$_a$ = ***a.request*** *with* $x^{t_{n+1}}_{a\to b}$

(11) **else** /* less than much time left */

(12) **if** ***a.propose*** ∈ *set-of-possible-next-actions* **then**

(13) *next-action*$_a$ = ***a.propose*** *with* $x^{t_{n+1}}_{a\to b}$

(14) **else** *next-action*$_a$ = ***a.counter-propose*** *with* $x^{t_{n+1}}_{a\to b}$

(15) **endif** /* for much time left*/

(16) **else** /* t_{n+3} is close to the deadline, no time for a proposal */

(17) **if** ***a.offer*** ∈ *set-of-possible-next-actions* **then**

(18) *next-action*$_a$ = ***a.offer*** *with* $x^{t_{n+1}}_{a\to b}$

(19) **else** *next-action*$_a$ = ***a.counter-offer*** *with* $x^{t_{n+1}}_{a\to b}$

(20) **endif** /*for time-for-proposal*/

end

Algorithm 8.9 Decisions when *not-acceptable* and when *close* distance

Agent *a* concedes in generating its response if there is *less-time-left*, *much-time-left* and *deadlock*(*3*) or *middle time left* and *deadlock*(*2*). Otherwise, agent *a* prefers to trade-off.

For a decision on the next action in the bilateral protocol, if there is much time left then agent *a* triggers the *requested* state. If there is still time for a proposal but not *much-time-left*, then agent *a* triggers the *proposed* state which may lead to an agreement since the distance to an acceptable set of issues is close. Otherwise, if there is no time left for bargaining, agent *a* makes an offer. Theory 8.9 is the theory for Algorithm 8.9 for the *not-acceptable-and-close-distance-process* process.

$$(\textit{less-time-left} \vee (\textit{much-time-left} \wedge \textit{deadlock}(3)) \vee (\textit{middle-time-left} \wedge \textit{deadlock}(2)) \leftrightarrow [a.\textit{concedes-generating-}\, x^{t_{n+1}}_{a\to b}]\, \textit{generated}(a)$$

$\neg$(*less-time-left* $\vee$ (*much-time-left* $\wedge$ *deadlock*(*3*)) $\vee$ (*middle-time-left* $\wedge$ *deadlock*(*2*)) $\leftrightarrow$ [*a.trade-offs-generating-* $x_{a\rightarrow b}^{t_{n+1}}$] *generated*(*a*)

(*time-for-proposal* $\wedge$ *much-time-left*) $\leftrightarrow$[(*requested*(*b*)?; *a.suggest*) $\cup$ (*proposed*(*b*)?; *a.request*)] *requested*(*a*)

(*time-for-proposal* $\wedge \neg$*much-time-left*) $\leftrightarrow$ [(*requested*(*b*)?; *a.propose*) $\cup$ (*proposed*(*b*)?; *a.counter-propose*)] *proposed*(*a*)

$\neg$*time-for-proposal* $\leftrightarrow$ [(*requested*(*b*)?; *a.offer*) $\cup$ (*proposed*(*b*)?; *a.counter-offer*)] *offered* (*a*)

Theory 8.9 Decisions when *not-acceptable* and when *close* distance

8.6 Implementation of a Bilateral Negotiation

The algorithm focuses on designing and implementing a bilateral negotiation between two agents with conflicting goals to find a mutually acceptable agreement over a set of issues. The assumptions, inputs, outputs, algorithms for data flow between processes and some interesting implementation details in Qu-Prolog are described.

8.6.1 Assumptions

The assumptions are realistic given two competitive negotiating agents.

- **Self-interested agents**
  Both agents are self-interested with possibly opposing goals.
- **Set of issues**
  The agents negotiate over a set of multiple issues such as price, quality, guaranties, terms and conditions. The set of issues is agreed beforehand.
- **Agent's preferences**
  An agent specifies its preferences for the values of a set of issues. These are the initial values for each issue when starting a negotiation. Usually the initial value of an issue lies on the border of the range of acceptability for an agent

and is sent in the first message. An agent also has to specify the direction of change in value they would like each issue to take i.e. a buyer would like the issue price to decrease (*down*) and the warranty to increase (*up*) whereas a seller would prefer the opposite.

- **Deadline**

  An agent declares its deadline, t_{max}, and the maximum acceptable number of rounds. The number of rounds constraints the number of times an agent is able to respond to its opponent. The number of rounds is included for dealing with limited resources and when generating a set of issues consumes part of those resources. If resources (other than time) are unlimited, then an agent can trivially declare a large number of rounds.

- **Communication medium**

  The simulation uses TCP/IP. Agents are represented by unique names on an IP host or domain name. An agent knows the name and location of its opponent. Privacy of information is assumed and no other agent can join, interfere in or corrupt a negotiation.

- **Protocol**

  All participants know the logical theory of the bilateral protocol.

- **Strategies**

  A rational agent makes decisions using strategies and mechanisms defined in section 8.2. These strategies are used to evaluate a set of issues and to respond to an opponent after generating a new set of issues and values. The algorithms in section 8.4 and 8.5 are used for deciding on the next action from the bilateral protocol.

8.6.2 Negotiation Cycle using a Responsive Mechanism

An agent computes the constant k_j^a from the initial values given by a user for a set of issues. An agent later uses k_j^a to generate sets of issues. See section 8.2.2 for how to calculate k_j^a, α_j^a from k_j^a and the values in a set of issues from α_j^a. Algorithm 8.10 defines a negotiation cycle for an agent using a responsive mechanism. An agent can either initiate the process or respond to an initiator.

Pre-conditions: current state is ¬*negotiating*, assumptions in section 8.6.1

Post-conditions: current state is *closed*

```
begin
(1)  if sending-agent then                    /*sending-agent starts the negotiation */
(2)       initialise-time-and- k_j^a
(3)       generate and send entry message;
(4)       update beliefs including about negotiation state;
(5)  endif
(6)  while ¬(current-state == closed) do
(7)       if sending-agent ∨ respondent then   /* next one to send a message */
(8)            generate message;
(9)            send message to opponent;
(10)           update beliefs about message and about negotiation state;
(11)      endif
(12)      if current-state == closed then
(13)           exit;
(14)      else                                 /* waiting for a message */
(15)           receive message;
(16)                if time-has-not-been-started then /*received an entry  message */
(17)                     initialise-time-and-compute-k_j^a
(18)                endif
(19)           update beliefs including about negotiation state;
(20)      endif
(21) endwhile
(22) exit;
end
```

Algorithm 8.10 Negotiation cycle for responsive mechanism

An agent either initiates a negotiation or responds to an initiator. Time is discreet. The procedure *initialise-time-and-*k_j^a sets t_{start} to 0 and an agent counts time from that point on. This procedure also computes the constant k_j^a using responsive tactics. An agent starting the negotiation generates and sends the initial message.

An agent stays in the *while* loop when the state of negotiation is *open*. A *sending-agent* may need to send 2 or more consecutive messages. A *respondent* agent has received a message and now needs to reply. If an agent is a *sending-agent* or a *respondent*, it generates and sends its message. It then updates its beliefs about the set of issues sent and the state of negotiation on receiving acknowledgements.

If the state is not *closed*, an agent waits for the next state transition depending on its role as respondent or sending-agent. On receiving a message, an agent checks whether it is an allowable entry action and if so starts its time and computes k_j^a. Then a receiver updates its beliefs with the set of issues received and the state of the negotiation. If the state is not *closed*, the *while* loop is re-entered. The whole algorithm iterates until the state is *closed*.

8.6.3 Negotiation cycle for a Deliberative Strategy

In the algorithm for a deliberative strategy, an agent notes the amount of time it and its opponent takes to generate a set of issues and to respond.

```
begin
(1)  if sending-agent then                    /* starting the negotiation */
(2)       start counting time;
(3)       generate and send entry message;
(4)       note time of sending
(5)       update beliefs including about negotiation state;
(6)  endif
(7)  while ¬(current-state == closed) do
(8)       if sending-agent ∨ respondent then      /* next one to send a message */
(9)            generate message;
(10)           send message to opponent;
(11)           note time of sending a message;    /* for calculating t_opp_gen
(12)           update beliefs about message and about negotiation state;
(13)      endif
(14)      if current-state== closed then
(15)           exit;
(16)      else receive message;
(17)      endif
(18)      if time-has-not-been-started then
(19)           start counting time
(20)      else
(21)           compute max t_opp_gen
(22)      endif
(23)      update beliefs including about negotiation state;
(24) endwhile
(25) exit;
end
```

Algorithm 8.11 Negotiation cycle for deliberative mechanism

When a *sending-agent* initiates a negotiation, it notes the starting and sending times, generates and sends the entry message.

In the loop while the state is *open*, a *sending-agent* or *respondent* generates and sends a message according to deliberative mechanisms. An agent records the sending time which will be used to compute its opponent's maximum response time. Then that agent updates its beliefs with the sent set of issues and the state of negotiation on receiving acknowledgement.

If the state is *closed*, then the process is terminated else an agent waits to receive a message. On receiving a message, an agent either initialises its time if it has not done so yet or calculates its opponent's maximum response time.

The agent then updates its beliefs about the received set of issues and the state of negotiation and re-enters the loop. An agent keeps sending and receiving messages in a *while* loop until the state is *closed*.

8.6.4 Some Implementation Details

The algorithm is implemented in Qu-Prolog, [Clark and al. 1998], which allows multi-threading and inter-process communication between Qu-Prolog threads running on the Internet. Each agent in a bilateral negotiation is executed as a separate Qu-Prolog application. An agent has one thread forked for processing messages from an opponent and knows the latter's identity. Real numbers are scaled to integers since Qu-Prolog does not support real numbers.

Input

An agent's user preferences about the range of acceptability for a set of issues are an input as Qu-Prolog files. An example of a preferences file for a seller is:

```
negotiating_issues([ (price,20), (del_time,5), (warranty,3), (quality,g) ]).
weight([ (price,5), (del_time,2), (warranty,2), (quality,1) ]).
opp_weight([ (price,4), (del_time,1), (warranty,2), (quality,3) ]).
min([ (price,10), (del_time,1), (warranty,0) ]).
max([ (price,25), (del_time,8), (warranty,5) ]).
direction([ (price,up), (del_time,up), (warranty,down) ]).
```

```
eval_function(quality,[(vb,100),(b,200),(m,600),(g,400),(vg,200),(ex,0)]).
round(50).
max_time(60).
```

Example 8.1 Preferences of a Seller

The list *negotiating_issues* contains (name-value) pairs of the issues that are under negotiation and their values. In the above list, the seller wants to negotiate over four issues – price, delivery time, warranty and quality, where the initial value of each issue is given.

The list *weight* associates each issue with their importance to the seller. This list is used by an evaluation function to compute the overall score of a set of issues. The seller here places more importance on the price.

The list *opp-weight* is used in a deliberative strategy to store what an agent believes about its opponent's associated weights over a set of issues. An agent uses this set for generating a response. If an agent has no indication of its opponent's preferences, it associates equal or almost equal weights for each issue.

The lists *min* and *max* include the range of acceptability values for quantitative issues, here price, warranty and delivery time. The list *direction* associates with each quantitative issue whether that agent prefers an increase or decrease in that issue. An increase is given by *up* and a decrease by *down*. For any qualitative issue, such as quality, *eval_function* associates with that issue a list of possible instantiations and the scores a user places on them. Here, quality can be *very bad, bad, medium, good, very good, excellent*. A seller places a high score of *medium* since it brings it a profit with reasonable production costs.

The predicate *round* specifies the maximum number of acceptable rounds in a negotiation process. It is used to indicate the level of resources available. The deadline for an agent is given by *max_time* in seconds.

Output

The simulation informs a user about the state transitions, current time and messages being sent and received. The output indicates which sub-state of *closed* the process terminates in. If an agreement is reached, then the set of issues agreed upon and its score are given.

The score of a set of issues

An evaluation function computes the overall score of a set of issues and helps in rating them and generating new contracts. An evaluation function for agent a, V_a, returns an integer in the range of [0, 10,000], which is equivalent to considering the first 4 decimal numbers of a real number between 0 and 1. If the function returns an out of range value, then the score takes the limit value (0 or 10,000) depending on the direction of the issue. In calculating the score for a qualitative issue, the preferences of a user includes an *eval_function* and the score of its value can be extracted.

Notable Procedures with a deliberative strategy as example

A negotiation starts with a procedure *start_negotiation* and then continues by iterating between procedures *action* and *process_messages*. In *start_negotiation*, an agent specifies its opponent, then sends an initial message to enter the process and records the starting time. The initial message is created using the preferences of a user and consists of a set of issues and an allowed action for a state transition.

A sending agent executes the procedure *action* with parameters a set of issues and a state-triggering action for sending to its opponent. After sending a message, it records the sending time. On receiving an acknowledgement, an agent updates its beliefs that the current state of negotiation. An agent records the last 10 messages sent.

A receiving agent waits for a message via the *process_messages* procedure and follows Algorithm 8.10 or Algorithm 8.11. It updates its beliefs about its opponent from the message received and keeps a list of received messages. Then, if the state is *open*, a receiving agent generates and sends its response via the procedures *choose_action* and *action*.

The procedure *choose_action* finds all the possible state-transition-actions according to the bilateral protocol and the current state. Then *generate_action* generates the set of issues an agent will send. An agent records the time spent in generating values for a set of issues. The tactic relative tit-for-tat is used as default tactic in the responsive mechanism.

8.7 Performance Analysis

The performance of a negotiation with a user's preferences and chosen strategies is analysed. There are two agents, a buyer and a seller. After inputting a user's preference file, each agent negotiates autonomously towards an agreement. This section gives five case studies for varying the deadline, strategy and initial values for the set of issues and weights. Let *B* denote a buyer agent, *S* a seller agent, *del* a deliberative mechanism and *resp* a responsive mechanism. The values in the result column, in the tables given below, are in the sequence *price*, *delivery time*, *warranty*, *quality*.

8.7.1 Case1: Varying deadline between 2 deliberative agents

Hypothesis 1: When two deliberative agents negotiate, the agent with the closer deadline gains less. The closer the deadline of both agents (the less resources), the less is the overall gain of the group.

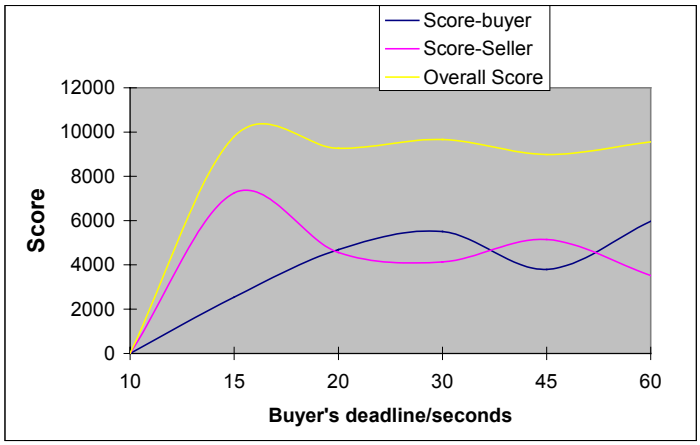
Experiment setup: One agent is a buyer and the other agent is a seller. Both the buyer and seller agents use deliberative strategies.

1. Both agents have the same deadlines which are equally decreased for both parties to investigate what is the overall gain.
2. The buyer and the seller interchangeably have a closer deadline than its opponent to see if their gain is affected by their role.
3. The difference between their deadlines is increased to analyse how their personal gain varies with such difference.
4. One agent starts with a very close deadline and it is checked if an agreement is reached.

Table 8.1 shows the effect of varying deadlines, *max_time*, on the set of issues agreed upon. The penultimate column shows which agent terminated the process and the last column gives the number of messages exchanged.

Row	Max_time		Result	Score			Last sender	#mes-sages
	Buyer	Seller		Buyer	Seller	Overall		
1	60	60	*agree*(18,7,4,ex)	5279	4779	10058	seller	10
2	60	60	*agree*(11,7,4,vg)	6543	2644	9187	buyer	12
3	45	45	*agree*(18,2,0,vg)	3794	5149	8943	seller	9
4	20	20	*agree*(14,6,1,vg)	4686	4558	9244	seller	5
5	30	30	*agree*(18,4,5,vg)	5508	3721	9229	buyer	7
6	30	30	*agree*(19,2,5,ex)	6126	3284	9410	buyer	6
7	60	30	*agree*(10,5,0,ex)	6097	3142	9239	seller	7
8	15	60	*agree*(20,7,0,vg)	2543	7244	9787	buyer	6
9	30	150	*agree*(19,8,4,ex)	4868	5400	10268	buyer	6
10	10	60	*timedout*	0	0	0	seller	5

Table 8.1 *Del-Del* varying deadlines



Graph 8.1 Scores against Buyer's deadline

Observation 1: From rows 1-6, the individual and group gains are affected by the agents' deadlines.

Conclusion 1: The scores of each agent and the overall score of the group are affected by their deadlines.

Observation 2a: Deliberative agents continue negotiating until their deadlines.

Observation 2b: Row 4 is an anomaly because with a closer deadline than row 3 the buyer manages to find a better deal, although the seller is worse off.

Observation 2c: Given the variation of deadlines with gains of rows 7 and 8 (i.e. twice more time left yielding nearly twice more gain for an agent in these rows), this proportionality is not repeated for row 9 where the two agents gain about the same even though the seller's deadline is 5 times further away. Row 9 is therefore another anomaly.

Conclusion 2: Although other factors are kept constant i.e. initial preferences and weights, the processing power of the machine on which the experiment onto may fluctuate. But such fluctuations are kept within a reasonable margin.

Observation 3: In row 8 of Table 8.1, the seller has 4 times more time left than the buyer and thus gains nearly 3 times as much in score. Same for rows 7 and 9.
Conclusion 3: An agent which has a closer deadline than its opponent concedes more and thus gains less than its opponent. So as to reach an agreement, an agent with a closer deadline moves towards its opponent's preferences.

Observation 4a: From rows 7-8, the gain of the agents is disproportional to the difference in the proximity of their deadlines.
Observation 4b: If one of the agent's deadline is very near, then it sends a take-it or leave-it offer as in row 8.
Conclusion 4: Although varying an agent's deadline yields a corresponding change in their gain, such change in gain may however be disproportional because once the agents have found an acceptable set of issues, their rate of concession does not depend so much on their deadline.

Observation 5a: There is a fair exchange when both agents have the same deadlines.
Observation 5b: However, in row 2, both agents have the same deadlines, but with more messages being sent.
Observation 5c: From rows 2, 5 and 6 the buyer sends the last offer and achieves a higher score. Same for the seller in row 3.
Conclusion 5: The results show that the last agent offering an agreement gains a higher score.

Observation 6: In Table 8.1, there is a significant intersection region between the range of acceptable values for each agent. During this negotiation, the agents find an acceptable contract quite early and negotiate to gain higher scores.

Observation 7: As soon as an agent receives an acceptable set of issues, it concedes less and lets its opponent approach it.
Conclusion 7: The agent receiving the first acceptable set of issues is usually the one gaining more in the end. It continues bargaining for maximising its benefits but with less pressure to reach the acceptable region, while its opponent compromises more in its responses.

The summary of the conclusions from case study 1 are:

- The agent with the closer deadline achieves less.
- When both agents have the same deadline, they are more likely to achieve the same score. The agent sending the last offer or reaching its acceptable region first stands more chance of gaining a higher score.
- Hypothesis 1 is supported.

8.7.2 Case2: Varying deadline between a deliberative and a responsive agent

Hypothesis 2: A deliberative agent achieves more than a responsive agent since its decision making is more complex.

Experiment setup: The negotiation between a deliberative and a responsive agent is investigated.

1. Both the responsive and deliberative agents have the same deadlines to see how their gain is varied with respect to only their strategies.
2. One of the agents has a closer deadline to compare their adaptability.

	Mechanism		Max_time		Result	Score			Last sender	#m
	Buyer	Seller	*B*	*S*		Buyer	Seller	Overall		
1	Del	Resp	60	60	*agree*(17,2,2,ex)	5458	3814	9272	buyer	14
2	Resp	Del	45	45	*agree*(16,8,0, g)	2868	6400	9268	seller	13
3	Del	Resp	45	45	*agree*(11,2,0,ex)	6258	2614	8872	buyer	14

4	Resp	Del	30	30	*agree*(12,8,0,m)	3336	5265	8601	seller	11
5	Del	Resp	30	30	*agree*(12,3,2,vg)	6051	2635	8686	buyer	10
6	Del	Resp	60	30	*reject*(9,3,3,ex)	0	0	0	buyer	12
7	Resp	Del	60	30	*agree*(14,3,3,m)	4715	3300	8015	seller	11
8	Del	Resp	30	60	*agree*(17,2,2,g)	4258	4214	8472	buyer	10
9	Resp	Del	30	60	*reject*(23,5,2,m)	0	0	0	seller	13

Table 8.2 *Del-Resp* varying deadlines

Observation 1: From rows 1-5, when both the responsive and deliberative agents have the same deadline, then the deliberative agent achieves a higher score than its opponent.

Observation 1b: A deliberative agent tries to gain more by being the last agent to make an offer. A responsive agent is more passive.

Conclusion 1: Hypothesis 2 is satisfied.

Observation 2: From row 9, when the responsive agent has less time than a deliberative agent, the negotiation ends in a *rejected* state.

Conclusion 2: The deliberative agent had more time available, was trading off and was proposing sets of issues unacceptable to the responsive agent.

Observation 3: From row 8, if the deliberative agent has less time than the responsive agent, then an agreement is still reached as opposed to observation 2 above.

Conclusion 3: The deliberative agent is willing to concede and an agreement is reached. When its deadline is near, the deliberative agent makes a large concession and sends an acceptable offer.

Observation 4: In a negotiation between a deliberative and a responsive agent, more messages are sent than between two deliberative agents.

Conclusion 4: This is because a responsive agent takes less time to evaluate and generate a set of issues than a deliberative agent. Thus a deliberative agent has more time to make tradeoffs than when dealing with another deliberative agent.

Conclusion 4b: A responsive strategy is not as time consuming as a deliberative strategy because calculating trade-offs requires time.

Observation 5: The overall scores between a responsive and deliberative agent is lower than between two deliberative agents.

Conclusion 5: A responsive agent misses those solutions that increase its opponent's gain without decreasing the responsive agent's gain.

Summary of conclusions from case 2:

- A deliberative agent gains more than a responsive agent when they have similar deadlines.
- A deliberative agent tends to be the agent making the last offer, instead of the responsive agent.

8.7.3 Case3: Comparing deliberative against responsive strategies

Hypothesis 3: Responsive strategies produce less gain than deliberative strategies.

Hypothesis 3b: Responsive strategies still find an agreement with close deadlines.

Experiment setup: Both agents have the same strategies. Their deadlines are varied. Table 8.3 compares negotiations between two deliberative agents and between two responsive agents under the same conditions and preferences.

	Mechanism		Max_time		Result	Score			Last sender	#m
	Buyer	Seller	*B*	*S*		Buyer	Seller	Overall		
1	Del	Del	60	60	*agree*(18,7,4,ex)	5279	4779	10058	seller	10
2	Del	Del	60	60	*agree*(11,7,4,vg)	6543	2644	9187	buyer	12
3	Del	Del	45	45	*agree*(18,2,0,vg)	3794	5149	8943	seller	9
4	Del	Del	20	20	*agree*(14,6,1,vg)	4686	4558	9244	seller	5
5	Del	Del	30	30	*agree*(18,4,5,vg)	5508	3721	9229	buyer	7
6	Del	Del	30	30	*agree*(19,2,5,ex)	6126	3284	9410	buyer	6
7	Resp	Resp	45	45	*agree*(16,7,3,g)	4143	5017	9160	seller	19
8	Resp	Resp	30	30	*agree*(15,7,2,g)	4034	4903	8937	seller	18
9	Resp	Resp	5	5	*agree*(14,5,1,m)	3620	4672	8301	buyer	14

Table 8.3 *Del-Del* against *Resp-Resp*

Observation 1: The responsive mechanism has a higher performance time and is computationally less expensive.

Conclusion 1: The responsive mechanism is faster.

Observation 2: Even though responsive agents send more messages, their score found is lower than between two deliberative agents.
Conclusion 2: Deliberative strategies achieve more than responsive strategies. Hypothesis 3 is supported.

Observation 3: The responsive mechanism is less time-dependent when the deadline is not close.
Observation 4: In row 9, even when the deadline is close, responsive agents find an agreement.
Conclusion 3: Hypothesis 3b is confirmed.

Overall conclusion:
Deliberative agents find higher scores and responsive agents find faster agreements.

8.7.4 Case4: Guessing an opponent's weights

Hypothesis 4a: Accurate modelling of an opponent's weights allows an agent to individually gain more.
Hypothesis 4b: A higher overall score is expected when the participants know each other's preferences, resulting in a solution close to an optimum deal.
Aim: To investigate whether it is better to guess or to assign average weights to an opponent's preferences.

Experiment setup: To generate its next response, a deliberative agent uses its beliefs about the preferences of its opponent. An agent's list *opp_weights* stores that agent's beliefs about its opponent's preferences for each issue. In this case study, both agents use a deliberative strategy and initially assign values to their list *opp_weights*. The performance is examined when

- the buyer and the seller know each other's preferences
- both agents do not know their opponent's preferences and give equal weights to all issues in the list *opp_weights*
- An agent gives random values to its opponent's weight

In rows 1 to 5 of Table 8.4, the deadline (t_{max}) of both agents is 60 seconds. In rows 6 to 9, the deadline (t_{max}) of both agents is 45 seconds

Row		weight	opp_ weight	Result	score	overall	last sender	#mesgs
1	**buyer**	(4,1,2,3)	(5,2,2,1)	*agree*(18,7,4,ex)	5279	10058	seller	10
	seller	(5,2,2,1)	(4,1,2,3)		4779			
2	**buyer**	(4,1,2,3)	(5,2,2,1)	*agree*(18,5,2,vg)	4165	9372	buyer	10
	seller	(5,2,2,1)	(4,1,2,3)		5207			
3	**buyer**	(4,1,2,3)	(2,3,3,2)	*agree*(14,3,1,ex)	6051	8886	buyer	12
	seller	(5,2,2,1)	(2,3,3,2)		2835			
4	**buyer**	(4,1,2,3)	(1,4,4,1)	*agree*(16,3,4,b)	3783	7553	seller	10
	seller	(5,2,2,1)	(1,4,4,1)		3770			
5	**buyer**	(4,1,2,3)	(1,4,4,1)	*reject*(5,6,4,b)	0	0	buyer	12
	seller	(5,2,2,1)	(1,4,4,1)		0			
6	**buyer**	(4,1,2,3)	(5,2,2,1)	*agree*(18,2,0,vg)	3794	8943	seller	9
	seller	(5,2,2,1)	(4,1,2,3)		5149			
7	**buyer**	(4,1,2,3)	(2,3,3,2)	*agree*(12,4,2,g)	5108	8229	buyer	10
	seller	(5,2,2,1)	(2,3,3,2)		3121			
8	**buyer**	(4,1,2,3)	(2,3,3,2)	*reject*(21,3,3,vg)	0	0	seller	9
	seller	(5,2,2,1)	(2,3,3,2)		0			
9	**buyer**	(4,1,2,3)	(1,4,4,1)	*agree*(15,6,5,g)	5254	7882	buyer	10
	seller	(5,2,2,1)	(1,4,4,1)		2628			

Table 8.4 *Del-Del* agents, varying opp_weights

From Table 8.4, the scores depend on the weights assigned to the issues in *opp_weights*.

Observation 1: From rows 1 and 2, the buyer and seller model each other's preferences correctly.
Conclusion 1: Their individual scores are higher than other rows (depending on who is the last agent to make an offer). Hypothesis 4a is confirmed.
Conclusion 2: When an agent is correct about *opp_weight* the overall score is higher. Hypothesis 4b is confirmed.

Observation 2: From rows 4, 5 and 8, when the agents randomly assign values to each other's scores, a rejection is reached or the gain is not as high as other rows.

Conclusion 3: An agent who does not know its opponent's preferences can make false assumptions about them. Agents who make wrong or random assumptions about their opponent's preferences achieve a medium score and risk rejections. The score is also affected by the deadline i.e. decreases when the deadline of both agents is reduced.
Conclusion 4: A rejection may then result (given close deadlines) because an agent does not move in the right direction towards its opponent.

Observation 5: Rows 3, 7 and 9 show that agreements with average values in *opp_weights* list.
Conclusion 5: When neutral and almost equal values are assigned, the overall score is decreased but an agreement is still reached.

Conclusions from case 4:

- The best deals are reached when the agents know or rightly deduce each other's preferences.
- When an agent does not know its opponent's preferences, random assignment leads to a decreased score.
- For a reasonable deal, it is better for an agent to give equal weights to the issues in its list *opp_weight*.

8.7.5 Case5: Varying initial values of deliberative agents

Hypothesis 5a: If the initial values of the two agents are close, then a better deal is obtained.

Experiment setup: In this case, both the buyer and seller are deliberative agents except for row 3 between a responsive and a deliberative agent. The deadline of both agents is 60 seconds. The scores reached with different initial values for each issue in an agent's preferences are compared.

Row	Agent	Stra-tegy	Initial values	Initial Score	Result	Score	overall	Last sender	#mesg
1	**buyer**	Del	(5,2,4,vg)	8858	*agree*(19,1,0,g)	3068	8468	buyer	10
	seller	Del	(25,7,1,b)	9114		5400			

2	**buyer**	Del	(5,2,4,vg)	8858	*agree*(13,2,3,ex)	6926	9010	buyer	8
	seller	Del	(20,6,3,b)	6358		2084			
3	**buyer**	Del	(5,2,4,vg)	8858	*agree*(11,2,4,vg)	7258	8472	Seller	9
	seller	Resp	(20,6,3,b)	6358		1214			
4	**buyer**	Del	(5,2,4,vg)	8858	*reject*(6,4,1,ex)	0	0	buyer	10
	seller	Del	(20,5,3,g)	5672		0			
5	**buyer**	Del	(10,4,3,g)	6383	*agree*(10,3,0,ex)	6383	8953	buyer	12
	seller	Del	(20,5,3,g)	5672		2570			

Table 8.5 *Del-Del* varying initial values of preferences, deadline 60 seconds

Observation 1: Initial values with a high overall score for the issues lead to an agreement with an equally high score.

Observation 2: A responsive agent fares less well than a deliberative agent when both agents start with an average set of issues.

Conclusion 1: An agent who specifies an initial set of issues more favourable to it, will gain more in the end, even when conceding.

Conclusion 2: The initial preferences influence the concession rate of an agent.

Observation 3: When the two agents have similar preferences as in row 5, the overall score is higher.

Conclusion 3: Hypothesis 5a is supported.

Observation 4: In row 3, the seller starts with a low set of values and a rejection happens.

Conclusion 4: A rejection is possible when one deliberative agent starts with an unfavourable set of issues while the other deliberative agent starts with a favourable set. The latter agent receiving an acceptable proposal early stops conceding and trades off instead.

Observation 5: From row 2 and 5, the buyer achieves more.

Conclusion 5: If both agents start with average sets, then the agent receiving the first acceptable proposal achieves the better score. Once it reaches its acceptable region, that agent stop conceding and uses tradeoffs mechanisms.

Conclusions from case 5:

- It pays more for an agent to start with a favourable set of issues than with an average set.
- The agent that receives the first acceptable proposal stops concedes and ends up gaining more.

8.7.6 Conclusions from Simulation

Different test cases are studied between two agents with different strategies, varying deadlines, set of issues and inputs. The preferences of an agent play a significant role in determining its final gain in a negotiation. Agents that declare initial set of issues favourable to them gain more.

A deliberative agent usually achieves more when its deadline is reasonable. A responsive agent achieves less but is faster in reaching an agreement. Deliberative agents knowing each other's preferences lead to the best solutions. In practice, competitive agents do not know their opponent's utilities. When an agent does not know its opponent's preferences, wrong assumptions give rise to poor performance and possibly no agreement. For a reasonable deal, it is better for an agent to assign equal weights to the issues in its list *opp_weight* rather than random weights.

The performance of an agent is sensitive to its deadline and can be prone to exploitation by its opponents. A deliberative agent can observe its opponent's behaviour. It can understand its opponent's concession rate and its preferences over a set of issues. An agent may then be able to predict its opponent's next action according to the protocol and history. This leads to a planning capacity in an agent where a partial or full path of actions towards a favourable goal can be predicted. An agent may choose a next action with a lower score but eventually leading to a more favourable agreement. As such the responsive and deliberative mechanisms are just one-step decision-making strategies, considering only the next action that will bring the highest score. Planning usually involves reasoning about two or more steps in the future. Although planning requires more computation and time, it can offer a better overall gain. The next sections consider path finding and planning agents.

8.8 Planning Agents

Agents can be endowed with AI techniques, such as search algorithms, for determining the sequence of actions in solving a problem, [Yokoo and Ishida 1999; Poole and al. 1998]. A set of potential partial solutions to a problem can be found. Search algorithms are used in resolving constraint satisfaction problems (definition of a possible solution), path-finding (a method of generating possible solutions) and game theoretic problems. Path-finding problems involve finding a set of moves from a source state to a goal state. Constraint satisfaction problems (CSP) require finding a goal configuration where as game theoretic problems (two-player games) deal with competitive scenarios. Game theory unrealistically tends to assume complete information or beliefs about an opponent's strategies. In standard search algorithms, an agent requires its global beliefs about a problem to perform each sequential step. This is not applicable when an agent has limited computational rationality and resources.

A plan may be regarded as an (partially or not) ordered set of paths, where the set is dynamically constructed and updated as a negotiation progresses. A planner is a problem solver that can produce plans (sequences of actions) to achieve some goal, [Poole and al. 1998]. The input to a planner is an initial world description, a protocol, a goal and strategies and the planner find a sequence of actions that will transform an initial world into one in which the goal is achieved. Forward planning, [Bacchus and Kabanza 1996], may be reduced to a path-finding problem and can use heuristic search strategies as in section 8.9.2. Backward planning involves backward chaining from a goal. The STRIPS planner, [Fikes and Nilsson 1971], regression, [Waldinger 1977], and partial order planning, [Weld 1994], aim for a smaller search space by considering only those actions that achieve a goal. In the STRIPS planner, sub-goals are ordered in case they become undone. Ordering is not always possible and re-achieving sub-goals wastes effort. A regression planner addresses the problem of interacting goals by keeping track of all the sub-goals. A partial-order planner only commits to ordering between actions when needed. A partial-order plan is a set of actions and their preconditions in a partial ordering showing precedence of actions.

The logical theory of a protocol may be considered as a directed state-space graph where a node is a state of negotiation, an edge is an atomic action and a path is a process. Search algorithms may be used to find a path towards a goal state. The following sections discuss planning from an ANML protocol through reasoning about the set of possible paths to a state.

8.8.1 The Tasks of a Shopping Scenario

Consider a shopping process between a retailer and a consumer. One can envisage a simple retail agent as seeking to maximise a long term profitability goal by wise buying, efficient distribution and stock management, and successful sales. In contrast, the goals of a retail customer are more subtle, involving, say maintenance of a personally adequate supply of garments, foods, or other consumables, through purchase within a fixed budget. The joint shopping process is illustrated in Figure 8.2 by an incomplete hierarchical JSD diagram [Jackson 1975], where task sequencing is left to right, iteration is loosely indicated by the symbol *, and shading distinguishes the activities of different or joint agents. Note that at the lowest level each task is attributed to a single agent, but it is not yet necessarily atomic.

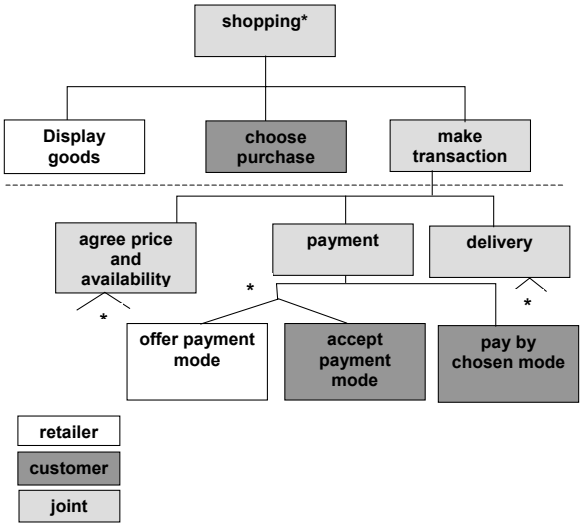


Figure 8.2 Shopping process between a retailer and a customer in JSD

The shopping process is a joint process which can be decomposed into abstract parts done by either one agent or both agents at a time. First the retail agent displays the goods being offered, perhaps in the form of a catalogue or an event sent on an event channel to broadcast what is proposed. The customer then

browses through the offers and chooses what he/she wants to purchase. At this stage there has not been any interaction between the two agents except for the sequencing itself, and synchronising this is a suppressed detail. The process of making a transaction will embody greater interaction between the two agents. In Figure 8.2, the *make transaction* joint process has been broken down into three joint processes. These three processes can themselves be further broken down into sequential actions done jointly by the two agents or by a single agent. The illustrated decomposition is by no means definite and is just one of many possible decompositions. The hierarchical shopping task as a joint process in ANML is as follows:

$$(retailer \cup customer).shopping = (r{:}retailer.display\_goods;\ c{:}customer.choose\_purchase;\ \{r, c\}.make\_transaction)^*$$

$$\{r : retailer, c : customer\}.\ make\_transaction = \{r, c\}.\ (agree\_price\_and\_availability\ ;\ payment\ ;\ delivery)$$

The first composition expresses the shopping process as a joint process between a retailer and a customer agent and consists of three sequential processes: the retailer *display goods*, the customer *choose purchase* and the joint *make_transaction* process. Although a faithful representation of the joint processes is implicit in the JSD diagram, it is an idealisation which ignores abnormal termination, such as when the customer makes no choice.

8.8.2 Joint States of the Shopping Process

The effects of the actions of the agents on the states of the shopping process can be represented them in a statechart, as a dual of Figure 8.2. Abnormal transitions can now be portrayed. Whereas Figure 8.2 represents joint tasks and process hierarchy, Figure 8.3 illustrates the hierarchy of joint states for the shopping process linked by the abstract *display goods, choose purchase* and *make_ transaction* tasks, [Cunningham and Paurobally 1999].

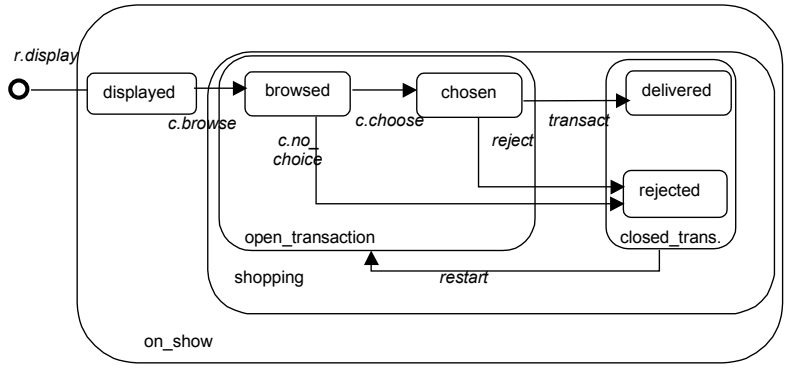


Figure 8.3 First Level State Transition Diagram of the Shopping Process

In Figure 8.3, the parent state *on_show* is the overall state. The *shopping* sub-state is entered by the customer browsing. The sub-states of *shopping* are the states *open_transaction* and *closed_transaction*. The states *rejected* and *delivered* are terminal sub-states that may be the sub-goals of an agent. Planning can lead to these goals and sub-goals. The *on_show* state is entered by the retailer making a *display*, leading to the *displayed* state. Then, the entrance of a customer through browsing leads to an *open_transaction*. Various state transitions change the sub-states of *open_transaction* and eventually terminate in a *closed_transaction* state. From the *closed_transaction* state, a customer or a retailer can restart a transaction. The process *transact* (*make_transaction* in Figure 8.2) may be a bilateral negotiation over the price, availability, the mode of payment and the delivery.

Each state in Figure 8.3 implies a partial state of affairs at an agent's side. For example, the *displayed* state infers a retailer having certain goods for sale. Similarly the *chosen* state of the shopping process implies a need of a customer and its monetary state. So the state transition diagram in Figure 8.3 represents the joint states of the shopping process but implies states of the involved agents. Theory 8.10 is a protocol in ANML for the shopping interaction in Figure 8.3 between a consumer, *c*, and a retailer, *r*.

shopping ↔ *one_of* (**[** *open_transaction, closed_transaction* **]**)
closed_transaction ↔ *one_of* (**[** *delivered* , *rejected* **]**)
open_transaction ↔ *one_of* (**[** *browsed, chosen* **]**)

$on\_show \leftrightarrow one\_of(\ [\ displayed, shopping\ ]\ )$

$\neg shopping \leftrightarrow none\_of(\ [\ open\_transaction,\ closed\_transaction\ ]\ )$

$\neg closed\_transaction \leftrightarrow none\_of(\ [\ delivered\ ,\ rejected\ ]\ )$

$\neg open\_transaction \leftrightarrow none\_of(\ [\ browsed, chosen\ ]\ )$

$\neg on\_show \leftrightarrow none\_of(\ [\ displayed, shopping\ ]\ )$

$\neg on\_show \leftrightarrow [r.display\ ]\ displayed$ (1)

$displayed \leftrightarrow [c.browse]\ browsed$ (2)

$browsed \leftrightarrow [c.no\_choice]\ \ rejected \vee [c.choose]\ chosen$ (3)

$chosen \leftrightarrow [r.reject \cup c.reject]\ rejected \vee [\{r\ ,\ c\}.transact]\ delivered$ (4)

$closed\_transaction \leftrightarrow [r.restart \cup c.restart]\ \ open\_transaction$ (5)

Theory 8.10 Protocol for a Shopping Interaction

8.8.3 The Paths of the Shopping Process

The ANML theory of the shopping process may be considered as a graph. Condition (1) gives the entry edges and conditions (2), (3), (4) and (5) are the internal edges of the graph. The states *rejected* and *delivered* are the end nodes. The *delivered* state may be a goal. Paths from an entry node to the goal node are derived. To do so, the '?' operator is used to test whether a path to a certain state is possible and to check if that goal state holds. A path to a state consists of processes leading to that state. For example the paths to an *open_transaction* state are the union of the following sets of paths:

- the customer entering the *open_transaction* through a *browse* action
- paths to the *closed_transaction* state following by the action of either the customer or the retailer restarting the transaction.
- paths to the *chosen* state

The paths to the *closed_transaction* state are inferred and may include the paths to an agent's goal. Let p_0 be paths to *open_transaction* and p_c be paths to *closed_transaction*. The paths p_0 and p_c can be derived from Theory 8.10.

$p_0 = p_c\ ;\ restart \cup (\ (\neg\ on\_show\ )?;\ r.display\ ;\ c.browse\ )$ (6)

$p_c = p_0\ ;\ ((\ chosen\ ?;\ \ \{c, r\}.\ transact \cup reject) \cup (browsed\ ?;\ (\ c.choose;$
$(\{c, r\}.\ transact \cup reject)) \cup c.no\_choice))$ (7)

Eliminating p_0 in (7) by substitution gives:

p_c = (p_c ; *restart* ∪ ((¬ *shopping*)?; *r.display* ; *c.browse*)) ; ((*chosen* ?; {*c*, *r*}. *transact* ∪ *reject*) ∪ (*browsed*?; (*c.choose* ; ({*c*, *r*}.*transact* ∪ *reject*) ∪ *c.no_choice*)) (8)

Let the path *entries* = (¬ *on_show*)?; *r.display ; c.browse* and the path *exits* = (*chosen* ?; {*c*, *r*}.*transact* ∪ *reject*) ∪ (*browsed* ?; *c.choose* ; ({*c*, *r*}.*transact* ∪ *reject*) ∪ *c.no_choice*). From (8), we can derive

p_c = (p_c ; *restart* ; *exits*) ∪ (*entries*; *exits*), whence: (9)

p_c = (*entries* ; *exits*) ∪ (*restart* ; *exits*)* (10)

A path to the *closed_transaction* state consists of entry to the *open_transaction* state followed by *exit* paths from that state with optional reiterations by restarting the transaction followed by *exit* paths. The recursive path equation (9) is solved as (10) in the regular algebra of paths, when viewing the logical theory of a protocol as a graph, by postulating the inference rule (x = (x ; A) ∪ B) ⇒ (x = B; A*). See [Backhouse and Carre 1973].

8.9 Finding Paths and Assigning Utilities

Negotiating agents can use the theories of protocols as inference engines in order to plan for a goal. From the ANML specification of a protocol, an abstract rule for a negotiation towards a path can emerge. For example, in the bilateral protocol, where the most abstract state is *negotiating*, let the abstract process, *negotiate* represent all the paths leading to a *closed* state of negotiation from a non-started negotiation. The axiom ¬*negotiating* ↔ [*negotiate*] *closed* holds in a bilateral negotiation.

The *negotiate* process contains all sub-paths that two agents can follow so as to reach their goal state. A goal-directed agent uses a sub-path of the process *negotiate* such that from an *open* state, an *agreed* state that satisfies their goal is reached. The formula ∃*p*:*path* <*open*?; *p*>*agreed* asserts that from an *open* state, there is a path *p* leading to the *agreed* state, a possible outcome of *negotiate*. The

paths may themselves be represented as ANML processes, inferred from the protocol, or may form part of the library of an agent's plans of actions. A rational agent will seek to derive a path that will lead it to an *agreed* state. An abstract negotiating state like *agreed* has significance for an agent because the *agreed* state represents a sub-goal of the agent, moving it closer to its greater goal.

Referring to the shopping process in section 8.8.2, the stages of offering and accepting a payment mode can follow a bilateral protocol. The retailer and customer follow a path in the abstract process *negotiate*. An example of a possible path for the joint process {*r:retailer,c:customer*}.*negotiate_payment* is:

r.propose_mode; (*c.request_mode* ;(*r.propose_mode* ; *c.request_mode*)* ;
r.offer_mode)$^{\leq 1}$; (*c.accept_mode* | *c.reject_mode*).

A protocol in ANML can be used as an inference engine to derive a valid sequence of actions from any source state to a target (not only a terminal) state i.e. a valid path. As an example of path derivation from the bilateral protocol, a sequence of actions can be derived leading from a *requested*(*Y*) state to an agreement between agents *X* and *Y*. For clarity, we give those rules of the bilateral protocol that are used for the derivation of the path $p_{req\text{-}agree}$, where *requested*(*Y*) $\rightarrow$ [$p_{req\text{-}agree}$]*agreed*.

$$proposed(X) \rightarrow offered(X) \quad (1)$$

$$requested(X) \leftrightarrow [Y.offer]\,(offered(Y) \wedge \neg proposed(X)) \vee [Y.propose]proposed(Y) \vee [Y.suggest]\,requested(Y) \wedge \neg(X = Y). \quad (2)$$

$$offered(X) \leftrightarrow [Y.agree]\,agreed(Y) \wedge \neg(X = Y). \quad (3)$$

$$proposed(X)) \leftrightarrow [Y.request]\,requested(Y) \wedge \neg(X = Y). \quad (4)$$

From rules (2) then (3), a possible path for agent *X* is *requested*(*Y*)?; *X.offer*; *Y.agree*

Applying rules (2), (2), (4), (4), (1) and (3) consecutively, another path $p_{req\text{-}agree}$ is *requested*(*Y*)?; *X.suggest*; *Y.propose*; *X.request*; *Y.propose*; *X.agree*

The '*' operator can be used in a path for repetitive actions. For example let *A* be either agents *X* or *Y* in the following path:

requested(*Y*)?; *iterative-suggestions* ; (*iterative-proposals* ∪ *A.offer*);*A.agree*

iterative-suggestions = (*X.suggest*; *Y.suggest* ∪ *Y.suggest*; *X.suggest*)*
iterative-proposals = ((*X.propose*; *Y.request*; *iterative-suggestions*) ∪
(*Y.propose*; *X.request; iterative-suggestions*))*

Consider a general aspect where a protocol can be used to find a path towards a state. Let the state, *outermost-state*, denote the overall parent state in a protocol e.g. *negotiating* in the bilateral protocol. All states are sub-states of the state *outermost-state*. Let all terminal states be sub-states of the terminal state *closed* in that protocol. If the protocol satisfies the termination property, then all paths are equal to or a sub-path of the path *all-possible-paths* where the formula ¬*outermost-state* ↔ [*all-possible-paths*] *closed* hold for that protocol. This rule is used for proving termination in a protocol.

If an agent has a goal $goal_x$, then its goal is achievable from the current state $state_x$ if $\exists\ p_x{:}path < state_x?;\ p_x > goal_x$. Goal $goal_x$ is possible from $state_x$ if there is a path p_x from $state_x$ to $goal_x$. For example, in an English auction, when $state_x$ is *auctioned* and $goal_x$ is *sold*, p_x denotes a possible path from the *auctioned* state to the *sold* state in the formula $\exists p_x{:}path <auctioned?;\ p_x>sold$.

8.9.1 Utility of Paths

Although, each agent has its own goal – to make a profit and to buy a service at a certain price respectively – when making a transaction, two agents will negotiate and converge to some final set of values that satisfies both goals, as in section 8.3. A rational agent will not only seek to achieve its goal of finding an *agreed* negotiation state. It will also try to achieve such a goal in a manner that will be most cost effective for it, so will negotiate with the sub-goal of closing the negotiation in an *agreed* state at the minimum cost. If the utility of a goal is defined as the difference between the worth of the goal and the cost of attaining the goal, and suppose that the worth to be the maximum cost that an agent is

willing to pay to achieve its goal, each agent may have a partially ordered set of goal utility values. Different agents can have differing utility orderings and mutual beliefs of each other's utility values may be incomplete. The agent goal may have the maximum utility, but its decisions can only be based on estimates of how to achieve this.

Earlier sections have studied an agent evaluating a set of issues and generating its response. An agent has to comply with the negotiation protocol and the external and internal constraints of the environment. An ANML protocol allows an agent to find a set of paths towards its goal. Therefore one step strategies can be extrapolated to plans of actions.

A path can be expressed in terms of other paths and sub-paths using ANML operators. An agent has to choose between the possible paths depending on its goals and preferences. The cost of a goal may be a fixed cost of the path to get to that goal or some estimated cost from a heuristic or statistical scheme. In order to choose between paths, build a plan of action, and eventually make its response, an agent can use assignments of estimated utility for each element of a path, then use some computational rationale to compose utility or estimated utility. A predicted set of possible paths can be ordered according to their utilities. For example, for a optimal choice of single path:

- *utility*(*path to state_a*) is *utility*(*state_a*) if *path to state_a* does not consist of subpaths
- *utility*(*path_a* ; *path_b*) is *utility*(*path_a*) + *utility*(*path_b*).
- *utility*(*path_a* ∪ *path_b*) is *maximum* (*utility*(*path_a*) , *utility* (*path_b*)).
- *utility*(ϵ), the null path, is utility of value 0.
- *utility*(ϕ), the empty set of paths, is utility of value - ∞.
- *utility*(*a**) is *maximum* (0 , *utility*(*a*) + *utility*(*a**)).

In such an idealised case, a full worth is assigned to desired (goal) states and zero worth to unsuccessful terminal states such as *rejected* in a shopping process. It may also be required that the cost of each transition to be positive in order that the utility of a loop is ultimately negative. However the actual cost of a transition

such as *transact* will include the price of the chosen goods, which depends on successful choice, while the success of the transact sub-negotiation may depend on this choice and the strategy of the retailer. Thus more realistic computations must replace the third and sixth utility axioms above by probability estimates for a set of paths, for example, the estimated utility of a set of terminating paths starting with *c.browse* and ending in a closed transaction, and thus incur more complex computations for statistical estimation.

An agent must thus have a library of negotiation plans, each providing a set of paths that enable it to interact with and respond to other agents with the aim of satisfying its goals. These plans may reflect the agent's individual beliefs about utility, the common beliefs of the agents about negotiation paths and estimates of the beliefs or behaviour of other agents.

8.9.2 A Discussion about Search Strategies for Path-finding

A search algorithm essentially consists of incrementally exploring paths from start nodes to goal nodes. A frontier between explored and yet to be explored nodes is maintained, where a search expands the frontier into the unexplored part of a graph. Search strategies determine the way in which the frontier is expanded and a graph is explored for possible paths. Traditional path finding may assign a cost with an arc or path of a graph. A negotiation can include can be the cost of performing a process or the cost (or worth/utility) of carrying out a goal. The cost of interacting is assumed to be negligible compared to the worth of a goal, which can influence path-finding strategies. See [Poole and al. 1998] and [Yokoo and Ishida 1999] for various search strategies and their time and space complexity.

Blind search strategies do not consider where the goal is and report success when they unintentionally find it. Examples of blind search strategies are depth-first strategies, breadth-first strategy, lowest-cost-first strategies. Their time and space complexity can rise exponentially with the risk of non-termination through infinite paths and cycles.

Heuristic search strategies expand those nodes that are the most promising for reaching a goal. A heuristic function $h(n)$ is an estimate of the path length from a

node n. There is a trade-off between the amount of work for deriving a heuristic value for a node and how accurately the heuristic value mirrors the actual path length, [Poole and al. 1998]. A heuristic function provides information about which neighbour of a node may lead to a goal. Examples of heuristic search algorithms are best-first search, heuristic depth-first search, *A** search.

*A** uses lowest-cost-first and best-first searches to consider both the current path cost and heuristic values in selecting the next node. For each path on the frontier, *A** uses an estimate of the total path length from a start node to a goal node along that path. If $g(n)$ is the cost (or length) of the path from a start node to node n, and $h(n)$ is an estimated path length from node n to the goal, then the total path cost from a source node (state) to a goal node (state) via n is $f(n) = g(n) + h(n)$. The next neighbouring node with the lowest $f(n)$ is chosen for expansion. *A** search always find a path and the first path found is optimal given finite branching and bounded costs. *A** search may be exponential in space and time.

Search strategies can be refined so as to reduce the search space, computation and time or to find the shortest path. For example, explicit cycle checking methods or multiple path pruning that do not explore neighbours that are already in the visited nodes so far avoid infinite paths and cycles. *A** search may use iterative deepening (IDA*) where the search is bound by $f(n)$ instead of depth.

To counter the exponential complexity of searching, search strategies can be enhanced with informed and intelligent methods to reduce space and time costs. The direction of search means whether starting from a source node to a goal or vice versa. In some situations the goal is not known and so restricts using backward search. Bi-directional search aims to reduce the search space by simultaneously searching forward from a source node and backward from a goal. A path is obtained when the two search frontiers intersect. Possible intersections between forward and backward search are considered as islands in the search graph. They are constrained to be on the path being searched for. The search problem is then decomposed into finding subsets of the path. However searching through a large space of possible islands may complicate the search. This decomposition relates to distributed problem solving or planning where partial or

abstract paths may be searched in the first instance before concentrating on detailed paths. Useful path decompositions or abstractions are required. There are other variants that can be used for optimisation of search strategies such as dynamic programming, hill climbing, simulated annealing, beam search and genetic algorithms.

In bilateral negotiations, two-player game strategies can be combined with heuristic search techniques for finding a set of possible paths. It is computationally expensive to generate a complete game tree for a complicated game. Game strategies for pruning the search space include the *minimax* and *alpha-beta* pruning procedure. In *minimax* strategy, [Shannon 1950], a player favours the move that will maximise its gain and minimise its opponent's gain. The evaluation value of each node, from the root node, can be defined recursively in terms of its child nodes. Alpha-beta pruning, [Knuth and Moore 1975], prunes out those branches that are not going to be visited in any case and saves computation in expanding them. See [Pearl 1984] and [Korf 1988] for more about path-finding, search strategies and two-player games.

8.9.3 The mental state of a planning agent

An agent can use its beliefs about the environment, its opponents, the negotiation and the mutual beliefs of a group to construct possible paths and to order and to choose between them for planning purposes. Chapter 7 provides some possible individual and mutual beliefs and knowledge of an agent that can help in its planning.

An agent may assign utilities to the possible paths, as in section 8.9.1, to obtain an ordered set of paths. Likewise, plans may be stored as an ordered set of predicted actions from a source state to a current state. Plans and the values assigned to them dynamically changes depending on the progress of a negotiation. An agent can then rate its plans for choosing which one to follow. The utility of a plan may depend on both its worth and the cost of carrying out the plan. For example, an agent may store information about its plans as follows:

{((*request*;*propose*;*request*; *request*; …;*offer*), 10), ((*request*; …;*offer*), 12), ….., ($plan_i$, $utility_i$, $probability_i$), ….}

The probability of a plan succeeding may be analysed according to a time frame, an agent's resources or an opponent's behaviour. Choosing a plan can reuse one-step decision making strategies where an agent concedes more as time elapses or resources gets scarce. At the beginning, an agent chooses a plan with high utility. During negotiation, if the agent faces significant risk of failure and losing out, it becomes more willing to choose plans of lesser utility but with more chance of succeeding.

There are uncertainties and risks in the outcomes when an agent has insufficient knowledge. Probability theory can be defined as the study of how knowledge affects belief, [Poole and al. 1998]. Possible worlds can be assigned with the probability of how likely the true state of affairs corresponds to a possible world. Reasoning under uncertainty involves an agent estimating what is most likely to happen and what may happen i.e. the probability of desired and undesired outcomes and the tradeoffs between them in choosing a path. Decision theory includes techniques for calculating the tradeoffs between the desirability of outcomes from sequences of actions related to the probability of the outcomes.

8.10 Summary

This chapter shows how to combine a protocol with strategies for a negotiation. One-step decision-making strategies are adapted for use with the bilateral protocol to evaluate and generate a set of issues and a state-triggering action. The behaviours and gains of an agent are analysed according to varying strategies. Path-finding and planning mechanisms are also discussed in the context of automated negotiation. Asymmetric and multiple-issue negotiations are assumed. However a multiple-issue negotiation can lead to an exponential growth in computation space and time. The attributes themselves need to be agreed upon.

9 Conclusion and Further Work

9.1 Summary

This thesis has addressed the need for precisely specifying and validating protocols and their properties, for negotiation between rational agents.

Automated negotiation saves time and cost, while exhibiting continuous, complex and dynamic problem-solving capabilities. A key aspect in automated negotiation is the rich interaction between agents, facilitated by languages, protocols, content languages and ontologies. Although protocols have become part of many multi-agent infrastructures, there is still a lack of methodologies for formalised protocol specification. Most of the methodologies used for this purpose, such as state transition diagrams, statecharts and AUML, are informal notations that are unsuitable for expressing agent interactions. This thesis shows that protocols expressed in them are prone to errors and ambiguities. In fact it argues that such methodologies are not expressive enough to fully represent open and dynamic multi-agent interactions. Thus more work is needed on the formal aspect of specifying protocols to facilitate the sharing and use of conversation policies by agents.

For a formalisation of interaction protocols and negotiation, this thesis related the processes in a negotiation to its states. It is difficult to a find notation where processes and states are given equal status, let alone form the basis for a simple and rational calculus for an executable system. There is no established notation to represent both the states and processes of an active agent, or a calculus for deciding why a particular negotiation should achieve the mutual goals. A suitably rich logic with action terms and variables appears capable of a practical reasoning system which can relate processes to goal states. Towards this end, the syntax and semantics of a meta-

language, called ANML, were specified based on extended propositional dynamic logic for representing and reasoning about agent interaction protocols. In ANML the state of a negotiation is related to the negotiation's sub-processes where that state is believed by a group of agents who propose state transitions. Being multi-modal, ANML inherits the properties of the PDL system. In addition the axioms and inference rules holding by virtue of ANML-specific connectors are defined.

Applications of ANML for fully and concisely specifying various negotiation protocols such as bilateral, multi-lateral, auctioning and promissory protocols were given. Interaction protocols were also be verified to avoid misunderstandings and ambiguities between participants. Protocols proposed in state charts and AUML were verified and the errors, incompleteness and ambiguities in them discussed. Each verified protocol was accompanied with a corrected version in ANML.

In addition to correctness, a protocol may be specified and analysed to exhibit desirable properties such as safety and liveness. These properties can be used to compare and choose between protocols. A number of safety and liveness properties were defined in ANML and proofs for them provided for the bilateral protocol.

Another aspect investigated in this thesis concerns the consistency of joint knowledge and beliefs between interacting agents. The assumptions and conditions for ensuring such consistency were given and proved in two cases – firstly in ensuring the consistency of joint knowledge in a group when the participants share and individually extend a protocol, and secondly for ensuring that a group of agents synchronously attains shared beliefs about the negotiation state, in an imperfect communication medium.

Finally a description of how goal-seeking agents can combine strategies with a protocol for a successful negotiation was discussed. Two strategies, deliberative and responsive, were adapted for one-step decision making in a bilateral negotiation and the results of the simulation analysed. The thesis also discussed how planning capacity in agents emerges from applying AI techniques such as path-finding, heuristic search or game theory to the state-space graph of a protocol.

9.2 Further Research

An extended formal analysis of the properties of the ANML formalism, such as consistency, completeness, soundness and decidability, and its implication represents an interesting avenue for further work: theorem proving methods are useful for conducting such analysis. It may be possible to translate ANML into a higher order logic and thereby exhibit the soundness property. Theorem proving also facilitates mechanisation of verifying the ANML theory of a protocol and its properties.

An interaction protocol can exhibit properties other than safety, liveness and those analysed in this thesis, for example game theoretic properties. Game theoretic properties may influence the private strategies of an agent and encourage behaviours like sincerity, social welfare, Pareto efficient, equilibrium and stable solutions.

Protocols can be represented using Petri nets, but such methodology does not support breaking a protocol into sub-protocols. In ANML, hierarchical states allow abstraction and nesting of protocols. Protocols expressed in other notations could also have been verified and the relation between μ-calculus, model-checking and ANML further investigated.

In synchronising the beliefs between agents about the state of a negotiation, rules of message exchanges and beliefs revision are given for a faulty communication medium. It is assumed that a message eventually gets through after persistent sending. Timeouts need to be incorporated in the solution to capture the case when an agent crashes or gives up.

A simulation on a larger scale and according to different negotiation protocols would allow exploring in more detail behaviours of agent in maximising gains. Adaptive protocols and strategies would be useful in dynamic, open environments and allow planning.

ANML and the theory of a protocol are based on multi-modal logic. These may not be understood by a non-logically inclined user who would prefer a graphical notation

such as AUML or statecharts. An improvement would be to make ANML accessible to a non-formal user.

9.2.1 Further Developments in Related Areas

This section presents multiple research areas where ANML can be applied to, extended for or contribute to.

ANML could be rendered more expressive depending on the domain or application under study. For example, the intersection operator could be added to express concurrent processes. As such ANML is found to be expressive enough to model *m-n* agent interactions.

The semantics of protocols represent an interesting area of future work. This includes exploring the roles of an agent and the meaning of a particular state and process name. States can embody different levels of commitment such as the state *requested* compared with *agreed*. Negotiation could help in sharing common ontologies between societies of agents.

The individual states of involved agents, instead of a negotiation state, and the changes in their states as a negotiation progresses could be investigated. A high-level view may enable modeling changes in an agent's individual state, thereby addressing concurrency issues and how to construct interaction protocols.

Interaction protocols need not be rigorous in their enforcement. Adaptive protocols are desired in a dynamic and open environment. Planning of protocols allows constructing more flexible protocols or interactions from the goals of a group of agents.

Different phases in an electronic transaction involve service brokerage, navigation, selection, negotiation, payment and customer retention. The relation between negotiation and these phases could be investigated. There is scope for specifying an abstract theory of a negotiation and extend phases before and after a negotiation for learning a user's preferences, setting up an interaction and executing commitments and engagements. Learning helps an agent to adapt its behaviour in a dynamic

environment. An agent can learn about its opponents, its user's preferences and adapt a protocol and strategies for changes in the environment. Experience about past and parallel negotiations could influence decision making. How interactions are characterised in terms of resources change, exchange, production, consumption and market mechanisms can be investigated.

Dialogues and argumentation could be used to design richer protocols of interaction. These may resolve conflicts in multi-agent systems. Agents could justify and persuade their opponents about their points of view and reach compromises that would not have been obvious when complying with an inflexible protocol. A protocol in ANML could be extended to include arguments such as threats, promises, explanations, querying and appeals.

The set of issues in a negotiation may be vague because the product is difficult to specify or the participants are unsure about their preferences. An agent does not know exactly what is a good deal and the acceptability range if any is dynamic. The relative importance of the attributes is not fixed at the start. Therefore there is scope for further work on the uncertainty aspect of a negotiation.

Other possibilities for further work relate to current issues in the agent oriented field such as establishing trust and enforcing sincerity of opponents. Practical aspects for executing a negotiation could include extending ANML protocols for more than just negotiation, for example for tasks-accomplishment, coorperation, coordination techniques or dialoguing in a team.

Negotiation is pervasive to real-world and electronic interactions. Currently our work is being applied to investigate service negotiation between agents located on mobile communication devices. Mobile communications involve low bandwidth, latency and loss of performance, providing a realistic setting in analysing the consistency of the joint beliefs of a group.

Appendix A – Normal Modal System

A.1 Syntax of Modal Logic

This appendix gives the semantics of modal and multi-modal systems and the rules and properties of a normal modal system. It may be used as a complement to chapter 3 where the meta-language ANML is specified. The rules and axioms of a normal modal system also apply in ANML.

Modal logic is the logic of necessity and possibility. Necessity is what is true at every possible world and possibility is what is true at some possible world. The syntax of propositional modal logic is as follows:

Atomic formulae:	$p \in \phi$
Formulae:	$A \in Fma(\phi)$
$A ::= p \mid \bot \mid A_1 \rightarrow A_2 \mid \Box A$	

Other connectives include the propositional logic operators (negation, disjunction, conjunction, equivalence, True) and $\Diamond$. The formula $\Diamond A$ is equivalent to $\neg \Box \neg A$. If the formula A is possible, then it is not the case that $\neg A$ holds in all accessible worlds.

Kripke was concerned with giving precise meaning to modalities of necessity and possibility. The truthfulness of a proposition depends on all truth assignments deemed possible and not only on one single truth assignment as in classical logic. Kripke, [Kripke 1963], characterised a proposition as being *necessarily* true if it holds in every possible world, while a proposition is *possibly* true if it holds in some possible world. The possible worlds are accessible through a binary relation from the current world, called an accessibility relation. Hintikka, [Hintikka 1969], proposed multiple accessibility relations, where each relation is attributed to a specific agent or world. This allows personal belief or knowledge if the relation is reflexive. A sentence $\Box A$ is true if and only if A is true at every possible world. Sentence $\Diamond A$ is true if A is true at some possible world.

A.2 Semantics of Modal Languages

A standard model is a structure $M = (W, R, V)$. *Worlds* is a set of possible worlds in that model. V represents an assignment of sets of possible worlds to atomic sentences i.e. the possible worlds where a formula holds, as an interpretation of the model. V: $p \rightarrow Pow(W)$ is a function with propositions, ϕ, as domain and powerset of W, $Pow(W)$, as range. V tells us at which worlds (if any) each propositional symbol is true.

$M, w \models p$ iff $w \in V(p)$, $p \in$ PROP

R is a binary relation on W, ($R \subseteq W \times W$). R is an accessibility relation between 2 worlds and is defined by the modality in the language. World w_1 is accessible to world w_2 means that w_2 is possible relative to world w_1 through the relation R. An accessibility relation associates the world where a modal formula holds to a possible world in the model. A modality holds in world w_1 if the consequence of that modality holds in other accessible worlds such as w_2, which is possible relative to w_1.

The truth conditions of modal formulas use the relation R of relative possibility where $\Box A$ is true at w_1 if and only if A is true at every world w_2 that is possible relative to w_1. The truth at every or some possible world relative to the given world is used to define a modality.

$M, w_1 \models \Box A$ iff for every w_2 in M such that $w_1 R w_2$, $M, w_2 \models A$

$M, w_1 \models \Diamond A$ iff for some w_2 in M such that $w_1 R w_2$, $M, w_2 \models A$

Relations can be any sort of binary relation on *Worlds*; no assumption is made concerning its content or structure. Therefore we can define the modalities in a multi-modal language through accessibility relations.

Let $\xi \vdash$ or $\vdash_\xi$ denote provability in the modal system ξ. Let M and N be models of ξ. $M, w \models \varphi$ where $w \in W$ means that φ is satisfied in M at world w i.e. $w \in V(\varphi)$. M is a model for φ, $M \models \varphi$ iff $\forall w_{w \in W}$ ($M, w \models \varphi$). M is a model for a set of formulae Γ, $M \models \Gamma$ iff $\forall \varphi_{\varphi \in \Gamma} (M \models \varphi)$. φ is satisfiable if $\exists M, w$ ($M, w \models \varphi$). φ is valid ($\models \varphi$) if $\forall M$ ($M \models \varphi$). Models M and N are modally equivalent (over a modal language L), if $\forall \varphi_{\varphi \in L} (M \models \varphi$ iff $N \models \varphi)$. M and N are isomorphic ($M = N$) if they are copies of the same model.

A.2.1 Multi-Modal Semantics

Given a set of modality labels *MOD*, the set of well-formed formulas φ of the multimodal language over propositions ϕ and *MOD* is

$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid <\alpha> \varphi \mid [\alpha] \varphi$

for all $p \in \phi$ and $\alpha \in MOD$. A Kripke model M is a triple (W, $\{R_\alpha \mid \alpha \in MOD\}$, V). W is a non-empty set of states or worlds, R_α is a binary relation on W with modality α, and V is a valuation function with

domain propositions ϕ and range powerset of W. V tells us at which states each propositional symbol is true.

$M, w \models [\alpha] A$ iff $\forall w^1 (w R_\alpha w^1 \rightarrow M, w^1 \models A)$

$M, w \models <\alpha> A$ iff $\exists w^1 (w R_\alpha w^1$ and $M, w^1 \models A)$

A.2.2 Propositional Dynamic Logic

The syntax of PDL programs denoted by α is $\alpha ::= \pi \mid \alpha_1 ; \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha^* \mid A?$

A model for propositional dynamic logic is a structure of the form, $M = (Worlds, \{R_\alpha: \alpha \in Prog(\phi, \Pi)\}, V)$. R_α is a binary relation on *Worlds* for each program α, and $M, s \models [\alpha] A$ iff $sR_\alpha t$ implies $M, t \models A$ where s and $t \in Worlds$. The relation, R_α, between two worlds, s and t, reflect the intended meanings of program α, which can itself consist atomic sub-programs. In world s, $[\alpha]A$ is true iff A holds in all possible worlds t, related to s via R_α. Program α is executed to change from state s to state t. A model M is defined to be standard if it satisfies the following conditions:

$R_{\alpha;\beta} = R_\alpha \circ R_\beta = \{(s, t, u) \in Worlds: \exists u(s R_\alpha u \ \& \ u R_\beta t)\}$

$R_{\alpha\cup\beta} = R_\alpha \cup R_\beta$

$R_{A?} = \{(w, w) \in W: M, w \models A\}$

$R_{\alpha^*} = R_{\alpha;\alpha^*} = (R_\alpha)^*$ = ancestral of R_α

A.3 Axioms for Normal Systems of Modal Logic

ANML encompasses the following axioms and rules for PDL and normal systems in modal logic.

$<\alpha>(\beta \vee \delta) \leftrightarrow <\alpha>\beta \vee <\alpha>\delta$

$[\alpha](\beta \vee \delta) \leftrightarrow [\alpha]\beta \vee [\alpha]\delta$

$[\alpha](\beta \wedge \delta) \leftrightarrow [\alpha]\beta \wedge [\alpha]\delta$

$<\alpha>(\beta \wedge \delta) \leftrightarrow <\alpha>\beta \wedge <\alpha>\delta$

$<\alpha>\beta \wedge [\alpha] \leftrightarrow <\alpha>(\beta \vee \delta)$

$[\alpha](\beta \rightarrow \delta) \rightarrow ([\alpha]\beta \rightarrow [\alpha]\delta)$

Monotonicity of $<\alpha>$ $\dfrac{a \rightarrow b}{<\alpha> a \rightarrow <\alpha> b}$

Monotonicity of $[\alpha]$ $\dfrac{a \rightarrow b}{[\alpha]a \rightarrow [\alpha]b}$

A.3.1 Axioms in PDL

$<\alpha\cup\beta> \delta \leftrightarrow <\alpha>\delta \ \vee <\beta>\delta$

$[\alpha\cup\beta] \delta \leftrightarrow [\alpha]\delta \ \wedge [\beta]\delta$

$<\alpha;\beta> \delta \leftrightarrow <\alpha><\beta>\delta$

$[\alpha;\beta] \delta \leftrightarrow [\alpha][\beta]\delta$

$<\alpha?>\beta \leftrightarrow (\alpha \wedge \beta)$

$[\alpha?]\beta \leftrightarrow (\alpha \rightarrow \beta)$

$[\alpha^*]\beta \rightarrow \beta$

$\beta \rightarrow <\alpha^*>\beta$

$[\alpha^*]\beta \rightarrow [\alpha]\,\beta$

$<\alpha^*>\beta \leftrightarrow <\alpha>\,\beta$

$[\alpha^*]\beta \leftrightarrow [\alpha^*\alpha^*]\,\beta$

$<\alpha^*>\beta \leftrightarrow <\alpha^*\alpha^*>\,\beta$

$[\alpha^*]\beta \leftrightarrow [\alpha^{**}]\,\beta$

$<\alpha^*>\beta \leftrightarrow <\alpha^{**}>\,\beta$

$[\alpha^*]\beta \leftrightarrow \beta \wedge [\alpha][\alpha^*]\,\beta$

$<\alpha^*>\beta \leftrightarrow \beta \vee <\alpha><\alpha^*>\,\beta$

$[\alpha^*]\beta \leftrightarrow \beta \wedge [\alpha^*]\,(\beta \rightarrow [\alpha]\beta)$

$<\alpha^*>\beta \leftrightarrow \beta \vee <\alpha^*>\,(\neg\beta \wedge <\alpha>\beta)$

A.4 Rules of Inference in a modal system

The rules of inference below apply in normal system in multi-modal logic over $Fma(\phi, \omega)$. Let formulae $A, B \in Fma(\phi, \omega)$, processes $\alpha, \beta, \rho \in Proc(\phi, \omega)$ and Σ a normal logic system in $Fma(\phi, \omega)$.

- **Modal axiom Df◇ (definition of ◇)**

  Df◇ $\quad <\alpha>A \leftrightarrow \neg([\alpha]\neg A)$

The formula $\neg([\alpha]\,\neg A)$ is read as it is not the case that in every possible world, the formula $\neg A$ holds after executing process α. If α consists of sub-processes, then $<\alpha>A$ means there is a possible path α to a world where A holds. There is a possible path to state A after the execution of α iff not all executions of α lead to $\neg A$.

- **RN, RM, RR and RE**

The inference rule *RN* states that if A is a tautology then A holds in all the worlds after the execution of any process α.

If A implies B and process α triggers A, we can infer from *RM* that α triggers B.

If A and B are sub-states of C, then from rule *RR* a process α that triggers A and B also triggers the parent state C. The rules *RN*, *RM*, *RR* and *RE* follow from *RK* for $n = 0$, 1, and 2, respectively.

RN. $\quad \dfrac{A}{[\alpha]A}$

RM $\quad \dfrac{A \rightarrow B}{[\alpha]\,A \rightarrow [\alpha]B}$

RR $\quad \dfrac{(A \wedge B) \rightarrow C}{\qquad}$

Appendix A

$$([\alpha] A \wedge [\alpha]B) \rightarrow [\alpha] C$$

RE is the rule of inference between equivalent states and allows to derive that the same process triggers equivalent states.

RE $\quad \dfrac{A \leftrightarrow B}{[\alpha]A \leftrightarrow [\alpha]B}$

Proof in system Σ:

$A \leftrightarrow B \quad = (B \rightarrow A) \wedge (A \rightarrow B)$ (by propositional logic)

$= ([\alpha]B \rightarrow [\alpha]A) \wedge ([\alpha]A \rightarrow [\alpha]B)$ (by RM)

$= [\alpha] A \leftrightarrow [\alpha]B$ (by propositional logic)

- **N, M and C axioms**

N. $[\alpha] \top$

Proof: By *PL*, $\vdash_\Sigma \top$. By RN, $\vdash_\Sigma [\alpha]\top$

M. $[\alpha](A \wedge B) \rightarrow ([\alpha]A \wedge [\alpha]B)$

Proof: From *PL* and *RM*.

By Propositional logic (*PL*) $\vdash_\Sigma (A \wedge B) \rightarrow A$ and $\vdash_\Sigma (A \wedge B) \rightarrow B$

By RM $\vdash_\Sigma [\alpha] (A \wedge B) \rightarrow [\alpha] A$ and $\vdash_\Sigma [\alpha] (A \wedge B) \rightarrow [\alpha] B$

By *PL* $[\alpha] (A \wedge B) \rightarrow [\alpha] A \wedge [\alpha] B$

C. $([\alpha]A \wedge [\alpha]B) \rightarrow [\alpha] (A \wedge B)$.

Proof: By *PL* $\vdash_\Sigma (A \wedge B) \rightarrow (A \wedge B)$.

By RR, $\vdash_\Sigma ([\alpha]A \wedge [\alpha]B) \rightarrow [\alpha](A \wedge B)$.

- **R and K axioms**

R. $[\alpha] (A \wedge B) \leftrightarrow ([\alpha] A \wedge [\alpha] B)$

Proof From M and above and introducing equivalence.

K $([\alpha] (A \rightarrow B)) \rightarrow ([\alpha] A \rightarrow [\alpha] B)$

Proof By *PL* $\vdash_\Sigma ((A \rightarrow B) \wedge A) \rightarrow B$

By RR $\vdash_\Sigma ([\alpha](A \rightarrow B) \wedge [\alpha] A) \rightarrow [\alpha] B$

By *PL* $(C \wedge D) \rightarrow F$ is $C \rightarrow (D \rightarrow F)$

By *PL* $([\alpha](A \rightarrow B)) \rightarrow ([\alpha] A \rightarrow [\alpha] B)$

A.5 Rules and Axioms for <α>

We give the rules and axioms underlying the possibility modality <α> holding in a normal system for modal logic. By the schema Df◇ <α>A is ¬([α] ¬A). <α>A has the modal meaning of possible worlds i.e. <α>A holds in world w_1 if after executing α, there is some world(s) accessible to w_1 where A holds. In the context of a graph or a negotiation, <α>A can be read as the possible path α to a goal state A.

- **RK◇**

RK◇ $\quad \dfrac{A \rightarrow (A^1 \vee \ldots\ldots \vee A^n)}{\langle\alpha\rangle A \rightarrow (\langle\alpha\rangle A^1 \vee \ldots. \vee \langle\alpha\rangle A^n)} \qquad (n \geqslant 0)$

RK◇ expresses that if there is a path to a sub-state, then there is a possible path to at least one of its parent states.

- **Rules RN◇, RM◇ and RR◇**

RN◇ $\quad \dfrac{\vDash \neg A}{\neg(\langle\alpha\rangle A)}$

RM◇ $\quad \dfrac{A \rightarrow B}{\langle\alpha\rangle A \rightarrow \langle\alpha\rangle B}$

RR◇ $\quad \dfrac{A \rightarrow (B \vee C)}{\langle\alpha\rangle A \rightarrow (\langle\alpha\rangle B \vee \langle\alpha\rangle C)}$

RN◇, RM◇, RR◇ follow from RK◇ for n = 0, 1, and 2, respectively. For RN◇, when n=0, the conditionals in RK◇ are identified with the negation of their antecedents, [Chellas 1980]. RN◇ says that if ¬A is a tautology then there is no possible path to A. RM◇ expresses that a possible path to a sub-state is a possible path to its parent state too. RR◇ similarly means that a possible path to a sub-state with 2 parent states is a possible path to at least one of its parent states.

- **RE ◇**

RE ◇ $\quad \dfrac{A \leftrightarrow B}{\langle\alpha\rangle A \leftrightarrow \langle\alpha\rangle B}$

Proof:

$A \leftrightarrow B$	$= (B \rightarrow A) \wedge (A \rightarrow B)$	(by propositional logic, *PL*)
	$= (\langle\alpha\rangle B \rightarrow \langle\alpha\rangle A) \wedge (\langle\alpha\rangle A \rightarrow \langle\alpha\rangle)$	(by RM◇)
	$= \langle\alpha\rangle A \leftrightarrow \langle\alpha\rangle B$	(by propositional logic, *PL*)

Appendix A

If A and B are mutually dependent sub-state and parent state (or are the same state), then a possible path to A implies a path to B and vice versa.

- **Df □**

  Df □ $[\alpha] A \leftrightarrow \neg (<\alpha>\neg A)$

  Proof: using *PL*, Df ◇ and RE

If all paths lead to A then it is not possible to get to a world where $\neg A$ holds. This schema is useful when striving for a goal e.g. an agreement.

- **Axioms N◇, M◇ and C◇**

  N◇ $\neg (<\alpha> \bot)$

  Proof: By *PL*, $\models \neg\bot$. By Rn◇, $\models \neg <\alpha> \bot$

  M◇ $(<\alpha>A \vee <\alpha>B) \rightarrow <\alpha>(A \vee B)$

  Proof: From *PL* and *RM*◇

A variant of M◇ would be $(<\alpha>A \vee <\rho>B) \rightarrow <\alpha \cup \rho>(A \vee B)$

- **Axioms C◇, R◇**

  C◇ $<\alpha>(A \vee B) \rightarrow <\alpha>A \vee <\alpha>B$.

  Proof: By *PL* $\vdash_{\Sigma} (A \vee B) \rightarrow (A \vee B)$.

  Hence, by RR◇, $\vdash_{\Sigma} <\alpha>(A \vee B) \rightarrow <\alpha>A \vee <\alpha>B)$.

  R◇ $<\alpha>(A \vee B) \leftrightarrow <\alpha>A \vee <\alpha>B$.

  Proof From C◇ and M◇, biconditional.

- **Axiom K◇**

  K◇ $(\neg(<\alpha> A) \wedge <\alpha>B) \rightarrow <\alpha > (\neg A \wedge B)$

  Proof From $B \rightarrow (A \vee(\neg A \wedge B))$ and RR◇

If α does not lead to A but to B, then executing it yields a world with $(\neg A \wedge B)$.

The above axioms and rules over $[\alpha]$ and $<\alpha>$ are those of a normal system as for modal logic and propositional dynamic logic.

Appendix B - Electronic Commerce and its Architectures

B.1 Introduction

Technology can now be used in new ways to offer original and innovative services which would not have been possible otherwise. In the last decade information and communication technologies (such as Internet and the World Wide Web) have given birth to Electronic Commerce. Electronic commerce is defined as performing commercial transactions on-line. Electronic Commerce offers several advantages over traditional ways of doing business. Organisations perceive Electronic Commerce as a means of offering better services that are accessible to a wide range of market participants. Secondary costs that occur during real life business transactions are eliminated in electronic commerce e.g. delivery of services, payment costs, credit costs or costs that are induced by the uncertainty of exchange rates. Organisations can thus gain competitive advantage. There are a growing number of organisations that are adopting online trading approaches to exploit the advantages of electronic commerce.

The first part of this appendix contains a survey of the evolution of electronic commerce with respect to its various aspects. Cunningham, Paurobally and al. [1998] provide a case study of an existing online service provider for electronic documents. Our findings allow us to list the requirements for electronic commerce. Finally we prioritise these requirements and describe a possible classification of the steps in an electronic transaction. The second part of this appendix presents the CORBA architecture, [OMG 2000], CORBA services and facilities, from [Orfali and Harkley 1997a]. It describes CORBA-compliant higher-level frameworks such as the business object level architecture. The third part of this appendix consists of CORBA-compliant architectures for electronic commerce. In particular we examine a negotiation framework based on CORBA and CORBA-services.

Part I

B.2 The Evolution of Electronic Commerce

We performed a study on electronic commerce in 1998, [OSM Consortium 1997], and some of the observations appear to be still valid. The main findings then were disappointing sales patterns due to lack of interactivity,

advertising space being hard to sell and proprietary technology being abandoned in favour of open standards. The conclusions suggested that technology had been unsatisfactory and that there was a need for a complete infrastructure for electronic commerce, [OSM Consortium 1997]. We found a limited number of vending patterns in the electronic markets. The commercialisation pattern was still nothing more than a copy of what non-commercial people think real life commerce is. Usually this is a Web page representing the shop window, along with some form of catalogue and a shopping basket. This is still the case today though the number of online purchases has increased.

Today most companies have a web presence and many provide an online shopping facility from which all participants can benefit from reduced costs and from location and time transparency. The latest statistics show that about 100.2 million people in the U.S. or nearly half of the adult population with access to the web have made a purchase online at one time or another, [Cox 2001]. An estimated $3.5 billion was spent online in March 2000 and $13.8 billion in 2000 for travel services, [Cox 2001].

However, the Internet is an interactive medium. The study did not find any sophisticated interactive process which demonstrated the full potential of the Internet and the realised benefits for electronic commerce, for example, in the way hypertext enriches information access. To this end, research is being performed on a range of associated issues such as brokering, [Sycara and al. 1997], security, [Tahara and al. 2001], trust, [Poslad and Calisti 2000], negotiation, [Jennings and Wooldrigdge 1997; Rosenschein and Zlotkin 1998; Sandholm 1995], ontologies, [Ontolinga; Mahalingam and Huhns 1997; Huhns and Singh 1997], multimedia, law, [Shoham and Tennenholtz 1995], and payment, [Wang and al. 1998; Yi and Okamoto 1998]. Current online shopping sites represent the first generation of e-commerce applications with client and server software and with humans still being involved at both ends, adding to transaction costs. To remove these costs and to automate time-consuming tasks, software agents that are personalised and continuously running may be used. For example, in auctions extending over several days, like Ebay, an agent knowing the preferences of its owner may watch out for interesting auctions and at any time strategically bid no further than a given reserve price. Specialised agents may be developed to support the different steps during a transaction, from searching and filtering large and unstructured online information to brokering, negotiation and paying.

If the consumer buying behaviour is divided in six main stages: need identification, product brokering, merchant brokering, negotiation, payment and delivery, as in [Guttman and al. 1998], then so far there are agent applications supporting the first three stages. Shopping assistants like PersonaLogic, [Guttman and al. 1998], and Firefly, [Maes and al. 1999], address the initial two stages and help customers find products through filtering and recommendation respectively. Other shopping assistants like Jango, [Jango], and Bargainfinder, [Krulwich 1996] are involved in merchant brokering, i.e. selecting providers. Bargainfinder performs on-line price comparisons by sending requests for price to different sellers while Jango as an advanced Bargainfinder masks the fact that those requests are coming from a central site and simulates them as genuine requests for competitive reasons. However all of these applications concentrate on price and unfortunately overlook possible value-added services offered by merchants.

B.3 Different Aspects of Electronic Commerce

The importance of payment systems

Security issues relating to payment are perceived as a determining factor for the proliferation of electronic commerce. Thus the main research efforts appeared to be concentrated on payment and security systems. To be able to show 'up front' that a secure technology for payment is being used is itself a sign of reliability. Yet behaviour studies do not show this element as a determining factor for first time purchase: the potential buyer usually goes all the way through the 'shopping process' and then decides whether the payment method offered by the vendor is acceptable or not. About 60% of the sites studied used the idea of a Virtual Caddy (Shopping Basket) for the shopping process and just under 90% accepted credit cards. Although roughly 50% of the selected commercial Web sites used encryption technology, it is commonly accepted that more than 50% of Web-based transactions happen in insecure environments, i.e. the buyers don't know or don't care. Interestingly, today most compensations in case of fraud are handled not by the banker or the customer but by the merchant.

A number of security protocols have been developed to enable secure electronic commerce transactions. Among these, Secure Sockets Layer (SSL) technology is used in many major web servers and browsers to secure communications over the Internet through encryption and digital certificates. Secure Electronic Transactions (SET) is a protocol that provides secure transmission of on-line payment information. Applications developed over Java can use the Gateway Security model of the Java Electronic Commerce Framework to enable access to sensitive data through only controlled gateways.

Emerging Standards and Systems

Open standards and protocols help lower the costs of offering new services on the customer desktop. Sun Microsystem's Electronic Commerce Framework extends the core Java platform for electronic commerce applications. The Java Commerce APIs support basic services to create new electronic commerce applications such as on-line shopping malls, home banking or electronic brokerage. A Java Commerce Client (JCC) permits electronic commerce transactions from the desktop. Users are provided with a Wallet-like user interface, a database, and an extensible platform that enables the use of a variety of payment instruments and protocols for a large number of e-commerce operations. Examples of other applications are Microsoft® BackOffice®, Microsoft Wallet and IBM's Managed Electronic Transaction Services.

Advertisements

About 60% of the sites studied used advertising. This takes place by advertising on other sites (referring), on one's own site or real-world advertising. Webmasters often use the opportunity to promote specific parts of their sites. It is common knowledge that referral links (advertisements which will take the user to a different site) are hard to sell. Only the major sites, in terms of traffic, are able to attract advertisements. Studies showed that two-thirds of the advertising spending went to the 10 largest sites. This indicates that advertising is not seen by the market to offer any real value except for a few exceptionally active sites. Even in those cases there is the problem that users don't often follow the links. This has led to a change in the pricing of web adverts. Advertisers are not willing to pay a price per person that has seen the advert. Instead they will only pay a certain price per person that has clicked on the advertisement and hence referred to the advertiser's site.

As most home users are reluctant to pay for some online services, advertising may increase in some areas. With new technology advertisers will be able to target the right portion of the market, for example, by advertising on a small site or channel with very few hits, but read by the right kind of people. Business users are usually quite willing to pay for useful information, so advertising may not be the only way to fund certain sites.

Legal Framework

The Internet is changing the way many organisations and individuals function. However, claims that a completely new legal framework is required to cover the 'electronic world' may be excessive. It is true that the current legal system lacks certain features, but principles that apply to the Internet can apply to many other areas. Widespread concerns with pornography and with copyright have elicited adaptation of pre-existing laws. On the other hand, there are aspects that do require new law. For example, if digital signatures are adopted and are deemed to be secure, laws need to be drafted to make them legally acceptable. Other matters can be raised too, such as security (anonymity and privacy mainly), trust and loss of government control. The issues involved here need clarification. Can the government be considered to be a trusted third party and do users think so?

Security

A simple error in e-commerce software can lead to large-scale fraud, theft, or breaking into the security and thus to the compromise of thousands of credit card numbers which can be quickly and widely distributed. With online commerce running to billions in the year 2001, new technologies are required to make electronic commerce both secure and user-friendly. Secure Sockets Layer (SSL) is the most widely used channel technology for securing communications over the Internet today. SSL uses public key cryptography and digital certificates to authenticate the Web site to the consumer and to ensure private communications by encrypting the channel. Many major web servers and browsers support SSL channel encryption.

Brokering

The Internet enables customers to locate and purchase products and services, with less dependence on their location relative to the vendor. The main difficulty is in locating the relevant products or services. This problem could be solved by special intermediary services, brokers to match vendors with consumers. As such, brokering services are still rare today in electronic commerce with a lack of adopted standards. The services that exist include search engines and appear to be business-to-business, not very interactive and have crude interfaces, or advertising at webmasters. Yet with brokering, customers would gain access to a larger market with more competitive prices. Vendors may not be so keen, because competition reduces profit margins, but in the long term, brokering should induce access to a global market as well as providing a market more responsive to changing customer needs.

CommerceNet is an industry consortium involved in accelerating the growth of Internet commerce and creating business opportunities for its members. Their motto is "to transform the net into a global electronic marketplace and make electronic commerce easy, trusted, and ubiquitous". It tries to unify vendors and end-users and help them jointly seize market opportunities. Agent toolkits for brokering include Kasbah, [Chavez and Maes1996], Tete-a-tete, [Guttman and al. 1998], Auctionbot [Wurman and al. 1998].

See Cunningham, Paurobally and al. [1998] for a case study of an industrial association providing information retrieval services to its customers. The case study helps us examine the requirements and constraints of brokerage-based business processes. The main finding was the importance of value-added information. Compared to search engines such as Lycos, AltaVista or Yahoo which have on-line indices running in hundreds of millions documents and which receive millions of queries daily, the business company's database captures only a fraction of these amounts. Yet, the value of the information stored is much higher, since the classification by human lectors assures a higher selectivity and quality. Business customers are willing to pay a high charge for comfort and for human expertise in both the processes of document acquisition and retrieval. Therefore, in an Internet-oriented setting, the integration of human resources may be an important factor within the preparation and provision of services.

B.4 Prioritising Requirements for Electronic Commerce

Although studies suggest that most requirements apply to any field of commerce, there are some sector-specific requirements. For example:

- Banking: Different laws on the protection of security of funds apply in different countries.
- Insurance: Complex mutual transactions which are negotiated and performed on-line need technical support (e.g. in the areas of reinsurance and co-assurance).
- Publishing: External parties with neutral judgements (e.g. rating services) are required in a decentralised environment to evaluate and recommend service providers for the quality of their services. Rating services may be useful in other areas too.

In table B.1, we have ranked the requirements that apply to most sectors. The rankings are based on the analysis of data collected from studies. The rankings are indicative and not absolute. The table shows the different criteria which are lacking currently in e-commerce architectures and the relative effort needed for supporting a criteria. Features are ranked by their perceived current level of support in the first column. Those that are best supported come first. In the second column, they are ranked according to the preferred support level. Finally, in the third column the ranking is by the effort required to reach the preferred level of support. Those features that are not currently well supported but are required are given first. They are followed by those features that are required less but might be fairly well supported.

Current level of support	Preferred level of support	Effort Required (adjusted)
1. On-line distribution	1. Certification	1. Auditing
2. Performance & Reliability	2. Payment	2. Negotiation
3. Catalogues	3. Performance & Reliability	3. Service Brokers
4. User Friendliness	4. Negotiation	4. Payment
5. Payment	5. Service Brokers	5. Certification
6. Privacy	6. User Friendliness	6. Privacy
7. Logs & Documentation	7. On-line distribution	7. Human Resource Integration
8. Human Resource Integration	8. Privacy	8. User Friendliness
9. Interoperability & Heterogeneity	9. Catalogues	9. Catalogues
10. Certification	10. Interoperability & Heterogeneity	10. Interoperability & Heterogeneity

11. Auditing	11. Auditing	11. Performance & Reliability
12. Service Brokers	12. Human Resource Integration	12. On-line distribution
13. Negotiation	13. Logs & Documentation	13. Logs & Documentation

Table B.1 List of requirements for Electronic Commerce

Security and certification - Currently, the most inhibiting factor for performing significant transactions is the lack of security. Security implies the concepts of privacy, authentication and non-repudiation. To achieve all of the above an infrastructure for security is required. The starting point for this infrastructure would be a way to identify individuals when needed. This could make them accountable for their actions and provide authentication and non-repudiation. Privacy is relatively straightforward to achieve. It is very likely that public key cryptography, [Rivest and al. 1978] will be the basis of security systems. The main problem of public key cryptography is the distribution of public keys. One must have a way of knowing that a certain public key belongs to a specific individual. A Certification Agency acting as a trusted third party can certify the ownership of public keys. An electronic commerce framework would necessarily need to support certification for these purposes.

Auditing Support - An architecture must have some support for audits in order to ensure that contracts are executed as set out in their terms and conditions. If necessary one should be able to check that an exchange was fair and that the legal formalities were taken care of. One way to do this would be through notary services.

Negotiation - Negotiations between humans are usually slow and may be tedious. This is especially true in complex situations. Currently there is only basic support for on-line negotiations mainly in the form of auction sites. The automation or partial automation of negotiations is required. The protocols and methods used need to be standardised to reduce costs.

Payment - One of the greatest requirements for electronic commerce is secure payment systems. The problem lies in the support of the payment systems and not the existence of payment systems as such. In other words, there is a lack of a framework which supports multiple payment systems. Within such a framework, consumers may negotiate with the vendor and agree on the preferred payment mechanism. Currently, there are various payment mechanisms, the most popular of which is payment with credit cards. Others have not yet reached their critical mass.

Human Resources Integration - Ideally, customers want to pay only for added-value services. In some cases it is possible to add value without human intervention. Nevertheless, the 'intelligence' of computers remains limited. It is necessary to integrate human resources or to make use of human expertise. The case study about *wf* shows that business customers would be willing to pay for human expertise. In any case, when computers are commonplace, the added-value will come mainly from humans. A framework for electronic commerce would have to be able to integrate different types of resources such as human and computer resources.

Brokers - Some types of on-line brokering already exist, for example Internet search engines, middle agents and institutions for managing auctions. We expect that vendors and consumers will realise the importance of brokering once payment and security issues have been resolved. Since the number of vending patterns on the

Internet is increasing, the need for brokering has started to be felt e.g. in the transport and logistics sector, mainly due to its decentralisation.

Catalogues - Electronic catalogues can provide complete and up-to-date information, greater choices, wider user access and on-line enrolment. Currently, electronic catalogues do exist, but they lack standardisation. As a result, each catalogue may require different hardware and software and have a different interface. Additionally they may not provide all the necessary functions and it may not be possible to integrate them with other catalogues. The product and service descriptions might be incompatible between catalogues. As catalogues would be the main way of displaying the profiles of different services, it is imperative that some standardisation takes place.

Interoperability - With the diverse range of existing and future systems, it is crucial that any electronic commerce framework is not platform specific. Adoption of standards such as those of the Object Management Group, Sun Systems or FIPA will ensure interoperability across a wide range of computer systems. An electronic commerce framework would be based on the idea of a global network; to achieve that, interoperability is a prerequisite.

B.5 Business Transactions

In view of the above requirements, an e-commerce architecture must provide the following services:

- **The distribution and inspection of information -** A business needs to establish a formal point of presence in the virtual world and to provide a catalogue of its services. These catalogues may be produced using any applicable tool. However, customers have to be able to inspect these service offers concurrently regardless of the specific tool used to publish them.
- **Negotiation -** In an open market, customers and providers both have ideas about what makes a good deal or a bad deal. Negotiation mechanisms allow a recursive interaction between consumer and deal maker in the resolution of a good deal. Solutions to a good deal are driven by independent business models of the respective parties. Negotiation supports convergence or migration between two points of view on what a service should contain and the contractual terms of engagement.
- **Contracts and commercial services -** Contracts provide a framework within which an agreement between multiple parties can be established, drafted and subsequently engaged. As such, contracts are conceptually shared structures that bind together a set of obligations such as the exchange of commercial services.
- **Customer relationship management -** Customer retention, service quality and competitiveness of on-line service delivery are key aspects of relationship management, involving long term interaction between a provider and the provider's customer base.

Present-day electronic commerce applications are limited chiefly to cataloguing on one side and payment facilities on the other with fixed vendor and client roles. The role of the vendor is passive after he/she publishes Web pages and is not able to negotiate with customers. Frameworks need to work at a level of sophistication closer to real world commerce than is evident. There is growing concern to solve the current lack of interactivity in online trading systems. Open standards and protocols help lower the costs of offering new services on the

customer desktop. Users, providers and software developers use interoperability standards to interact with components in different domains.

- Software developers and technology providers need a reference architecture to use for the development of their products. The architecture must be standardised enough to allow them to implement parts of an overall electronic market system and at the same time flexible enough to allow the market infrastructure to evolve.
- Consumers need a wider choice of products and services at competitive prices and a market that will adapt quickly to their needs. They should be able to use standard and non-standard services, as well as compound services to suit their individual requirements.
- Providers need the means to access a large, global market at very low cost, 24 hours a day, to offer services efficiently and effectively and to shorten their product cycle times. Providers need to be able to publish their services without the need to formally standardise them.

Electronic commerce systems should, [OSM Consortium 1997]:

1. Allow the decision-maker access to relevant information wherever it is situated.
2. Allow the decision-maker to request and obtain information management services from other departments in the organisations.
3. Proactively identify and deliver timely, relevant information that may not have been explicitly asked for.
4. Inform the decision-maker of changes that have been made elsewhere in the business process, which impinge upon the current decision context.
5. Identify the parties who may be interested in the outcome and results of the decision-making activity.

The characteristics in business processes are maximisation of profits; physically distributed, decentralised ownership; autonomous groups; natural concurrency; monitor and manage the overall business which is dynamic and unpredictable. Such commercial business transactions can be modelled in ways that encourage standardisation. For example, we can separate the construction from the execution phases of transactions. In the construction phase, a provider or consumer builds a commercial service in the following steps:

- Information Collection – the use of catalogues and brokerage systems to find on-line agencies, commercial services, service providers, consumers and third parties.
- Agreement– the construction of a contract, where the terms and conditions of engagement are agreed by the parties through negotiation and other mechanisms. This may involve negotiation to construct a contract and associated obligations.
- Engagement – the progression from agreement to commitment. In the real world, this is characterised by the signing of a document by all parties of a contract.

The execution phase consists of three other steps:

- Configuration – the framework over which electronic transactions take place, including the use of federated policies for security, payment, etc.
- Service Execution – the execution of commercial services in the context of higher level contract policies (like fair exchange).

- Termination – the validation and the closing of the contract across all participants, in accordance with policies concerning delivery assurance, fitness for purpose and associated obligations.

B.6 Conclusions on Requirements for E-Commerce

Studies show that the full potential of the Internet as an interactive medium for conducting online transactions is not being exploited. Electronic commerce is currently restricted to information collection with a shopping basket. Security, implying privacy, authentication and non-repudiation, are important requirements for electronic commerce. The security of an underlying infrastructure will determine its adoption. Two other important requirements are negotiation and brokering, covering a range of functionalities such as dialogues, matchmaking, effective information search and the supervision, drawing up and execution of contracts. Such services are currently absent in the online world. We have passed the first stages where electronic commerce may only be an extension of traditional commerce. A framework must be sufficiently standardised for practical use but be generic enough to support market evolution and emerging business models. An electronic commerce infrastructure needs to be interoperable with and make full reuse of the different technologies available. An emerging technology of electronic commerce has to coexist and evolve with more traditional information technology in business, where the existence of monolithic, legacy systems ill-suited to distributed access and management provides considerable inertia for change. In the real world there are several layers of interaction that can take place between a potential vendor and client, as well as several layers of intermediaries who can act or interfere. This is an area where distributed object architectures for interoperable components and business agent architectures have a growing impact.

Part II

Applications in large networks such as the Internet and the World Wide Web form a distributed environment through which information and resources are passed. Components may be aggregated using open systems and client/server interoperation standards to form higher level systems. In this part, we describe object oriented frameworks specified by the Object Management Group, OMG, which can support electronic commerce applications. We also describe a roadmap proposed by OMG for electronic commerce stemming from a European Commission project, OSM. We provide more details of CORBA services, facilities and CORBA compliant business architectures. While we mostly analyse CORBA and related applications, other framework embodying distributed object concepts are Microsoft DCOM and Java beans/OJDBC. DCOM being Microsoft's standard is subject to controversies. Java ODBC is relatively new compared to CORBA.

B.7 Object Management Architecture

One of the main contributions of the OMG towards distributed frameworks is the Common Object Request Broker Architecture, CORBA. The CORBA specification describes the Object Management Architecture (OMA) which consists of an Object Model and a Reference Model. The Object Model describes the structure of CORBA compliant objects and their terminology. An object is an encapsulated entity with an identity whose services can be accessed only through known interfaces by client requests, [OMG]. These interfaces are defined

in CORBA's interface definition language (IDL). IDL is a declarative language that allows objects to hide their implementation and only declare their interfaces. A client does not need to access the implementation of a server but rather it sends messages to the server object. The latter interprets the message and executes the service and may return the results to the client or cause an exception.

The server declares its interface as a set of attributes and of operations that a client can request. An operation has a signature that describes the parameters to be passed and returned, the exceptions raised and possible contextual information. An object can also support multiple interfaces through inheritance. The requests issued from client requests may have parameters which are used to pass data to the target object. An object implementation consists of the code for creating an object instance and the code for published methods which may update the system's state.

The OMA reference model defines the OMG components for distributed object oriented services. The Object Request Broker (ORB) is the core of the OMA and allows communication between clients and objects. The ORB acts as a bus and forwards requests from a client to a registered server and then forwards any reply from the server to the client. When using the ORB, an object does not need to specify the communication mechanism or the location of other objects. There are four categories of object interfaces that use the ORB bus as shown in Figure B.1. Each of the components in these categories can be used to further electronic commerce applications.

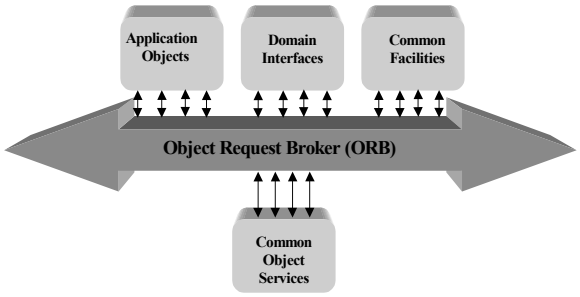


Figure B. 1: The OMA Reference Model

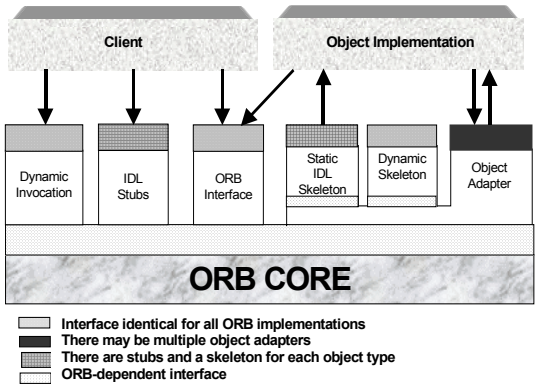


Figure B. 2: CORBA Architecture

- Common Object Services is a set of interfaces that can be called upon by distributed object programs. CORBA services such as security, transaction and session can be used in electronic commerce applications.
- Common Facilities are frameworks defined in IDL and are oriented towards end-user applications. They tend to define interfaces for the collaboration of application objects.
- Domain Interfaces provide a set of interfaces similar to the common object services and common facilities but applied to particular application domains.
- Application Objects are the client and server objects and applications. They use the above interfaces and the ORB. The interfaces are application-specific and not standardised.

B.8 CORBA

The Common Object Request Broker Architecture (CORBA), Figure B.2, promulgated by the OMG, specifies an open standard for client/server middleware. In this section, we describe the CORBA specification which can

form the basis for electronic commerce infrastructures. Objects can be both clients and servers. The interfaces of CORBA objects are defined in IDL. The IDL interface of a server can be considered as a contract of the methods provided by the server. IDL is a declarative language and its grammar is a subset of C++ with additional keywords to support distributed concepts. An IDL interface may include a component's attributes, supported methods with input and output parameters and their types, parent classes, exceptions and events that can be sent. IDL provides no implementation details so that IDL-specified methods can be written in and invoked in any language that supports CORBA bindings. A server's interface is added to an interface repository which can be accessed at run-time. A client accesses the interface repository to learn about the server. A client must know the object reference (identity) of the server and the operation to invoke along with the required parameters.

Figure B.2 shows the CORBA architecture. It shows the ORB and the interfaces of its component with respect to clients and servers. The Object Request Broker (ORB) is the core of the architecture and allows communication between clients and objects. The ORB acts as a bus and forwards requests from a client to a registered server and then forwards any reply from the server to the client. When using the ORB, an object does not need to specify the communication mechanism or the location of other objects.

There are two ways that a client can use the ORB to invoke a method on a server – by calling stub routines on the ORB statically or by dynamically constructing a request. On receiving a client request, the ORB finds the appropriate server object implementation, passes on the parameters and invokes the method(s). The ORB invokes a method on a server either statically through an IDL skeleton or through a dynamic skeleton. When the server has carried out the request, the ORB transfers back the result and control to the client. A client does not need to know the location of a server, the latter's implementation details or operating system. The server on its side does not distinguish between static or dynamic method invocation.

The Dynamic Invocation Interface (DII) – This interface allows dynamic construction of object invocations. CORBA has an API for looking up the interfaces of a server. It helps to generate the parameters and to issue a call after specifying the object and the operation to be called.

The Client IDL Stubs, Static Invocation – The client defines in IDL the services it wants to invoke. The IDL compiler generates IDL stubs that consist of code to invoke client services through the ORB and to perform parameter marshalling. A client has an IDL stub for each interface it uses on the server.

The ORB Interface – The ORB interface is the same for all ORBs. This interface consists of a few operations for both client and server implementations such as converting an object reference to a string and vice versa. The ORB core is responsible for locating a server object adapter and transmiting an invocation and its parameters. It then transfers control to the server implementation through the server skeleton.

Static IDL Skeleton – The server declares which methods, for each object, are implemented through this interface. Static IDL skeletons are generated by the compiler from declared IDL interfaces. The ORB calls the

implementation through these static skeletons. The skeletons do not know whether the client invocation is through the IDL stubs or the DII.

Dynamic Skeleton Interface (DSI) – Servers may have components that do not have IDL compiled skeletons. The implementation code provides descriptions of operations and their parameters to the ORB which itself provides the input parameters and the method call. The DSI finds the target object and method from the incoming invocation. Results from the implementation execution are passed back to the ORB. DSI are useful for implementing bridges between ORBs.

Object Adapter – An object adapter provides ORB services to object implementations e.g. generation and assignment of object references, activation and deactivation of server objects, security of interactions, method invocation and registration of implementations. The object adapter can also accept requests on behalf of the server's objects. CORBA 2.0 specifies that each ORB must support a standard adapter called the Basic Object Adapter (BOA). An ORB provider can support more specific services in addition to the BOA. The next version of CORBA defines the Portable Object Adaptor (POA) with backward compatibatibility with BOA. A POA allows programmers to construct object implementations that are portable between different ORB products. Since server objects are not running all the time but rather on demand from client requests, POA ensures the transparent activation of objects. POA also supports persistency such that an object's states can be saved beyond the lifetime of the server.

Interface Repository – This repository contains IDL-defined interfaces of the objects registered with the ORB. It is regarded as a run-time distributed database. The API allows access, store and update operations on the registered component interfaces and their method declarations. The information obtained may be used to perform requests at run time.

Implementation Repository – This repository contains information about the classes a server supports, instantiated objects and their object references. The ORB locates and activates implementations of objects through operations on the Implementation Repository. A client is able to call any object instance that implements the registered interface, [Orfali and al. 1997b]. In static invocation, an IDL pre-compiler generates static IDL stubs and the client knows, at compile time, what methods are to be invoked on which servers. In dynamic invocation, the client creates a request at run-time with an object reference, method and its associated parameters to invoke methods for which there is no IDL stub. A server does not distinguish between static or dynamic method invocations.

Static invocations are easier to program, by just invoking the method on an object and passing required parameters. Type checking is enforced at compile time in static invocation. Dynamic invocation offers flexibility and allows creation of requests on the run and adding new classes without changing a client's code. New services can be discovered at runtime such that generic clients are written with dynamic APIs and become more specific later at runtime. On the other hand, static invocations have better performance than dynamic invocation.

B.9 Java and Distributed objects

Java applets are copied to a client machine after a request from the client browser. Then the bytecodes are run through a verifier to safeguard the client from viruses before being executed by an interpreter. Java is portable because bytecodes can be loaded on any platform which runs a java virtual machine. The portability of Java applications renders transparent installation and porting of programs. The portability and error management of Java can be combined with CORBA's interoperability. Once the applets are downloaded on the client side, operations can be invoked on remote objects through CORBA. CORBA compliant applications written in Java are able to run on any platform and interoperate with other objects independent of their location and language. An application can be partitioned to run on clients and servers without recompiling with any update cascading from the servers to the clients. A Java applet acts as a client and invokes CORBA server objects.

B.10 CORBA Services

CORBA services are a collection of interfaces offering basic functions for using and adding features to objects. These services are used to construct distributed applications and are independent of application domains. Each CORBA service provides a specific function and can be combined with other services. We briefly outline the function of each CORBA service and suggest its relevancy with electronic commerce. Unfortunately many Object Services specified by the OMG still lack implementation products.

- Naming Service – The naming service maps names to unique object references and allows objects on an ORB to locate others. This function can be used in catalogue support and in brokering to find clients and providers.
- Life Cycle – The life cycle interface provides operations for creating, deleting, copying and moving objects while keeping the relationships between the objects consistent. Life cycle functions may used to create the entities involved in electronic commerce such as those representing participants, facilities and products.
- Events – An event is an occurrence in an object and is communicated to another object via notification messages. In a transaction, an object can either be a supplier or a consumer of events, both of which can initiate an event transfer. A consumer of events registers its interest in specific events and on event occurrence the supplier of events sends it notification messages. An event channel is an object on the ORB where suppliers send their events and consumers are notified about the events. Multiple suppliers can thus communicate with multiple consumers asynchronously without knowing each other.
- Object Trader – Service providers advertise their services with the trader by passing an object reference, its service type and properties. The trader maintains a repository of service types which consumers use to discover services that match their needs, their constraints and preferences. Traders from different domains can create federations and advertise their services in a pool.
- Transactions – A transaction is a contract binding the client to one or more servers and can be initiated by a client's request in the form of a set of method invocations.
- Concurrency – This service provides interfaces to acquire and release locks that let multiple clients coordinate their access to shared resources like catalogues and transaction states.
- Security – Details of a transaction must be kept private as well as a server's resources must be protected against unauthorised access. The ORB provides security for objects and allows them to be ported across

environments with different security mechanisms. Clients must comply with security requirements and provide identification, authentication and credentials. Resources are protected by access control lists and via encryption, audit trails, authorisation and non-repudiation policies.

- Persistent Object Service – Persistence means that the objects maintain their state in a non-volatile datastore after the process terminates or is interrupted. There are operations for retaining and managing the state.
- Externalisation – This service defines interfaces for externalising an object to a stream and internalising an object from a stream. On externalisation, the object can be transported to a different process, machine or ORB and then internalised for execution.
- Query – Queries can be performed to find objects or collections whose attributes meet the specified search criteria. A query service can delegate queries and combine the results for a global search.
- Collections – Objects can be grouped into collections with operations for querying, adding, replacing, removing and retrieving elements.
- Object relationships – This service allows to dynamically create and manage relations between objects and assign roles to objects.
- Time – The Time service helps to synchronise time using global time servers. The interfaces ascertain the order in which events occur, generate time-based events and compute the interval between two events.
- Licensing – The Licensing Service provides a mechanism for producers to control and charge the use of their intellectual property.
- Properties – From this service, named attributes (properties) can dynamically be associated with encapsulated components.

B.11 CORBA Facilities

CORBA facilities are IDL frameworks that provide services to application objects and aiming to support end-user applications. The facilities are divided into two categories: Horizontal Common Facilities, which are used by most systems, and Vertical Market Facilities, which are domain-specific. The Vertical Market Facilities represent technology that supports various market segments such as health care, retailing, manufacturing and financial systems. Horizontal Common Facilities mostly provide functions shared by systems. There are four domains:

1. The user interface facilities make an application accessible to its users and responsive to their needs.
2. Information Management facilities cover the modelling, definition, storage, retrieval, management and interchange of information.
3. System Management facilities cover the management of complex, multi-vendor information systems by service providers.
4. Task Management facilities support the automation of work, both user processes and system processes. The agent facility provides support for static and dynamic agents. The rule management facility supports knowledge acquisition, maintenance, and execution of rule based objects, such as intelligent agents.

B.12 Higher Level Frameworks: The Business Object Facility

The Business Object Component Architecture (BOCA) and the Business Object Facility (BOF) Interoperability Specification were two proposals to the OMG's for modelling business objects. These two specifications were

finally replaced by the task/session facility, which contains the key ideas from the BOF. The BOCA specification's main contribution was in a meta-model for describing business models, that is, it provided the necessary tools, concepts and language with which to describe a model of a business system. The BOF specification defines a CORBA compliant object in a business domain called a business object e.g. an employee. A business object has attributes and operations, but unlike in CORBA, operations have pre and post conditions enforcing their behaviour. It may also take on roles within a business and have relationships with other business objects. The BOF specifies a Component Definition Language (CDL), an extension of IDL, to describe business objects. The specified interfaces, protocols and shared services provide technical interoperability between business objects. The aim is for business objects to be defined and implemented in terms of business concepts without effort on the technical aspects.

A.12.1 Business System Domain

A business system domain (BSD) is a distributed objects system in a particular business domain. A consistent, recoverable representation of that business domain is maintained. The BSD may be as small as a solution to a particular problem, or as large as the information for an entire enterprise. Relationships between business objects should not cross BSD boundaries. However different business system domains may interoperate by loosely coupling through adaptors and achieve compatibility and integrity within each BSD.

A BSD may include address spaces that represent a vendor's framework. The latter may itself incorporate different business object types. Each business object type has a type manager that provides methods for accessing business objects. Each address space is under its own business object framework. Interoperating business objects may be in address spaces with different framework implementations. A federation of several BSDs is accomplished through the use of adaptors. An adaptor allows a business object to be represented in one domain, *A*, and linked to its primary representation in another domain, *B*, where its shared state and behaviour are implemented. The domain containing the adaptor is dependent upon the referenced domain which contains the primary representation of the shared object. The object in the primary domain should work independent of the adaptor in the dependent domain.

B.13 Advantages of CORBA

Along with along with the advantages of object-oriented programming, CORBA provides a high level approach to developing distributed applications.

- CORBA is more sophisticated than Remote Procedure Call (RPC) or database stored procedures. CORBA objects can be accessed remotely, independent of operating system differences.
- Since the implementation is separated from the interface, the client does not need to be aware of implementation details and of updates such as creation, installation and communication. Objects can be found based upon their interface, host machine or object reference, facilitating distribution, interoperability and heterogeneity.
- CORBA provides language-neutral data types and so allows invocation of server objects by clients written in any high level language that support CORBA bindings.

- Servers must register their IDL interfaces with the interface repository. Clients can use metadata to discover the servers and find how to invoke services at run-time. This makes CORBA a self-describing system with automatic generation of stub code through IDL mapping compiler.
- An ORB can run alone on a system or be connected to other ORBs using CORBA 2.0's Internet Inter-ORB Protocol (IIOP) services. An ORB can broker inter-object calls within a single process, multiple processes running within the same machine, or multiple processes running across networks and operating systems. This allows objects to interoperate through distributed systems.
- Legacy systems can be encapsulated through CORBA IDL by registering an interface for existing systems and making them appear as objects.
- ORB method invocations are specific. A function invoked on an object will return different results as when invoked on another instance, depending on the type and state of the object.
- CORBA has defined a set of services and facilities e.g. for creating and deleting objects, persistency, associations and security. A user can create an ordinary object and then make it transactional, secure, lockable, and persistent through multiply-inheritance from CORBA services, [Orfali and al. 1997]. CORBA offers facilities such as automatic server activation and deactivation, load balancing and some garbage collect mechanism.
- Combined with Java's portability, downloaded Java applets can interoperate with other distributed objects through IIOP routing. This frees web servers of managing extra communication.

B.14 Shortcomings of CORBA

- **Lack of formalisation -** One of the main difficulties encountered by CORBA users is understanding and using the framework. Most of them have a limited understanding of the CORBA architecture, as its specification is not clear. Using CORBA methodology can result in an overly complex system design with many similar fine-grain components to control and with a lack of component management mechanisms. CORBA has a steep learning curve requiring investments in time, new training and new architecture. A simple and formal CORBA specification with associated semantics is needed to clarify the standard. The preconditions and postconditions of a component, its functions and events have to be formally stated. Hence we need an overall abstract theory of the CORBA system.

- IDL is an **inflexible** component definition mechanism, which may be appropriate for generic low-level component communication but not for more context-sensitive components found at a higher level in the client/server hierarchy.

- **No holistic view for complex systems -** CORBA lacks the necessary scope to relate to a holistic view of the strategic relationship between organisations and their information systems as it is more relevant to information systems infrastructure, [Vinosky 1997]. CORBA is not suitable for modelling information flow in the organisation. It fails to provide sufficient conceptual basis for modelling complex systems as the individual objects and methods are too primitive and the design patterns are too generic and too rigid. One solution would be to capture purposeful behaviour in domain terms and allow flexible interactions.

- There is **minimal support for organising collectives** and no facilities for aggregation of objects to offer a variety of services. There is a need for richer structures to capture wider range of relationships for team building. CORBA does not specify a rich set of message types for co-operation of intelligent components.

- **Need for a complete development process -** A more complete development process than what CORBA provides is needed. A balance must be reached between applicability and reusability. There is no standard component-oriented development methodology to develop 'pluggable' CORBA components that will slot easily into application architectures designed to solve real-world business problems. Many specified CORBA object services still lack as implementation products.

- The **scope of the system testing is limited** by the component level i.e. components may be only tested as 'black boxes'.

- The **caller determines when to call a method** on an object and this places all the work on the caller. This is not feasible in a competitive environment where the callee might not want a competitor to access its services. The callee needs to be able to control who is invoking its methods. The caller and the callee may want to negotiate with each other and the CORBA framework on its own does not support this.

- **Autonomy -** Objects are obedient and their functionalities need to be programmed beforehand. The user has to provide choices and monitor the execution of the objects. They can not choose between actions autonomously and anticipate future trends. There is a lack of autonomy and ability to co-operate with other objects automatically.

- Too many vendor variations cause **compatibility problems**.
- **Extra download time**. To be able to communicate with another CORBA object, an applet needs an ORB to talk to and so an orblet is downloaded along with the classes that form the applet. Orblets are Java classes.
- Implementations are **immature and continually evolving**.
- **Limited mainstream acceptance** (DCOM and Java ODBC competition).
- Many **ORBs do not provide full functionality** of the CORBA specification.
- **Interoperability between different ORB** implementations can be a problem – code written for one ORB may need modification for use with another ORB.
- **Performance** can be slow.
- **Legacy systems** with large objects are complicated.
- **CORBA remains unproven**, slow to evolve, hard to administer.
- There is no **inheritance for exceptions**. Inheritance causes problems in versioning, so objects cannot support two versions of the same interface. IDL does not support overloading of operations.
- **Scalability** can be an issue if design is not well thought out.
- The **limited concurrency** model means there is no standard for thread priority, deadlines and timeouts.

In a dynamic environment, the object-oriented approach on its own fails, as there are no innate solutions for dealing with a situated problem solver and there is no mechanism for the objects to respond to changes in the environment autonomously and rationally. To solve this, higher level frameworks may be built on CORBA using its interoperation capabilities in order to provide more dynamic and complex behaviour. Such higher level CORBA-compliant frameworks can themselves be used to support electronic commerce applications. In the next part, we describe CORBA compliant architectures for electronic commerce.

Part III

B.15 EC Architectures - The Task/Session Facility

The Task/Session Facility, [OMG Task/session], defines business level notions of users, places, resources, tasks and workspaces as processes. It is specified in UML representing the business objects and relationships in the end user view of a distributed system. When instantiated, these CORBA compliant objects become configurations of people in places, using resources in tasks and processes. The WWW Document Object Model (DOM) level specification is used to complement the OMG task/session specification so as to handle data in XML structures. The task/session framework provides applications with business objects customised enough to be distinguished at the user end interface. These entities are able to inherit from tasks, users, desktops and workspaces and be linked to each other. The main types introduced in the task/session specification are:

- AbstractPerson contains information about people, a party or an organisation.
- User objects identify people and determine access to resources. Users have tasks and resources located in workspaces on a desktop and are connected or disconnected to a session.
- Workspaces are created by users and represent private and shared places where resources, including task and session objects may be contained.
- Desktop links users to workspaces. Each user has one desktop and many workspaces which may themselves be shared amongst users.
- Tasks describe user units of work that bind a user to selected data and process resources.
- Resources are collected in workspaces and represent process resources such as applications or data resources such as files or other CORBA objects.
- Usage is a relationship that connects tasks to processes and information resources.
- Containment is a relationship linking resources to workspaces.

B.16 Open Service Model Reference Architecture

Current electronic commerce applications, such as those on the World Wide Web, primarily support the information collection step. The OSM framework's objective is to enable the exchange of information between sophisticated mediators (brokers), consumers and providers of services during the construction and execution phases. OSM's mission statement is to provide an open service model for global information brokerage and distribution. Electronic brokers offer value-added services to deal with the vast amount of information available which would otherwise be difficult, if not impossible, to search manually. Brokering agents overcome the

limitations of direct negotiations between customers and suppliers through matchmaking, negotiating and monitoring transactions.

Information can be gathered from different sources e.g. an independent consumer association, a previous buyer of the product or special rating services. A broker may then selectively direct information to customers by maintaining databases about products and services. Electronic brokering cuts down on expensive searches. In the agreement and execution step, brokers may act as third parties and assist in negotiations. They maintain the anonymity of the participants during negotiations by acting as intermediaries and thus enable competitors to negotiate. Brokers may also ensure that transactions are carried out as agreed and that the consumers and providers complete their part of the deal.

The OSM Reference Architecture has been adopted by the OMG as a roadmap for electronic commerce. The architecture is based on CORBA 2.0 and discusses payment and security facilities, client interaction with service providers, service access, catalogue maintenance, negotiation and monitoring of service delivery, intermediate brokers supporting standard and non-standard services, intelligent service matching and dynamic service construction. Implementation of the OSM architecture used the BOF specification. The OSM project assumes that business objects developed under the system will be exposed through a generic browser which can represent collaborative or competitive services. The OSM architecture, Figure B. 3, is composed of three principal layers:

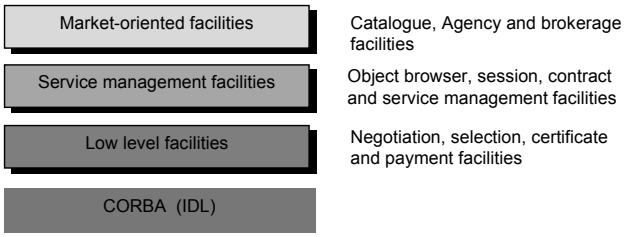


Figure B. 3: OSM Reference Architecture

Session facilities support asynchronous event communications by opening a channel between participants and providing transactional integrity and channel security. Providers advertise their products on relevant information channels and authorised consumers subscribe to channels of interest. Catalogues enable information browsing and inspection through their interface. Collections of items from different suppliers can also be constructed into a catalogue, so enabling brokerage. An agency facility establishes a formal query point and public query interface about a provider in an electronic marketplace.

The brokerage facility allows users to advertise and discover resources while ensuring anonymity and privacy. Participants are more focused in dealing with information about commercial services. The selection facility supports selection and configuration of supporting appliances and policies. The negotiation facility allows parties to communicate and mutually agree on a transaction. A negotiation process is defined as the progressive and controlled disclosure of information between parties leading to mutual agreement and subsequent engagement to that agreement. This engagement can be said to be contractually binding. The payment facility enables interactions between a buyer and a seller and any necessary third parties for the successful electronic exchange of goods or services. The value exchanged is governed by regulations and business requirements which are expressed in a "payment protocol" associated with each payment type.

The object browser provides a framework for presentation and management of components such as services, contracts, and certificates. It offers portability and interoperability of products and services across desktop environments. The Object Browser is expected to interact with a number of objects such as service descriptions and catalogues referred to as resources. The OSM object browser was implemented in JDK 1.2 beta version 3, with components built using the Java Swing classes. The object browser contains one or more view panels, each associated with a resource.

B.17 Negotiation Facility Specification

Parties in a business transaction may have differing goals and negotiation mechanisms are required to bring convergence between the two points of view and subsequent engagement in a deal. The OSM+ negotiation facility, [OSM SARL 1998], addresses the convergence, agreement and engagement phases of commercial transactions. It specifies a set of interfaces to support negotiation between two or more parties. Negotiation processes can be divided in three categories. In technical negotiation, the participants agree on technical applications such as method of payment or the negotiation of a value to be assigned to a property. In business level agreement, negotiation is on business factors like the subject of negotiation or the attribution of results. Promissory commitments deal with long-term agreements like promises to perform a process, rights and obligations between participants.

The negotiation facility is built upon the OMG CORBA 2.3 specification and the CORBA Object Services (COS). It reuses the Document Object Model, [DOM], and the Task/Session specification. The framework itself consists of the Community Framework and the Collaboration framework. Negotiation processes following business models run on the overall negotiation framework. Processes acting as customers and providers interact at a high level where the underlying layers in the framework provide a web of coordination interfaces for negotiation. At the highest level, lower level connectivity and interoperability functions are abstracted away to obtain a system where the objects can be viewed as communicating processes. These entities inherit from the underlying interfaces and have the capability to carry out negotiation with other participants and converge to an agreement that satisfies the user's goals. Furthermore they can be given capabilities to enter into meaningful coordination and thereby become trustworthy enough to enter into engagement procedures. Finally processes draw up contracts and ensure execution and delivery.

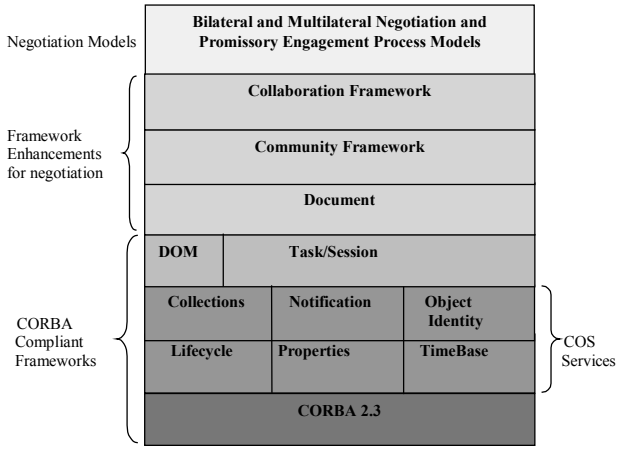


Figure B. 4: Negotiation Facility Framework

CORBA 2.3 as the lowest layer provides interfaces for clients to invoke operations on remote servers. The CORBA layer provides networking, method invocation and interoperability between objects, and higher up the hierarchy, between negotiating processes. The COS services layer provides interfaces for the management of objects, of their properties and relationships and of time. The naming service (identity) allows objects on an ORB to locate other objects. The Notification service enables objects to register or deregister their interest in specific events before sending notification on event occurrence. Offers, requests, rejections and acceptance can be thought essentially as events that are passed to negotiating processes.

The Negotiation facility builds extensively on the Task/Session Specification described in section B.15. The WWW Document Object Model (DOM) level specification complements the OMG task/session specification to handle data in XML structures by providing a common interface to XML and HTML documents. The community framework of the negotiation facility extends the notion of workspace by introducing membership roles between users and workspaces. Communities of users emerge from this layer. Users are associated into groups following membership constraints. The interfaces at this level are divided into those supporting membership management and those defining a community, an agency or an enterprise. A community interface offers client applications controlled access to resources that may be required during the course of a collaboration. Agency inherits from Community and LegalEntity types to introduce the notion of legal community such as a company that maintains jurisdiction of a set of resources. A Membership interface enables associations of users of the type Member according to rules exposed under a MembershipKind. The operations in Membership allow addition, listing and removing of members, querying Member participation and exposing the state of the Membership. There is an attribute in the MembershipKind interface to qualify the extent of information disclosure and membership kind to public or private parties.

The collaboration framework introduces interfaces to support the execution of collaborative processes amongst communities of users. These collaboration processes can be customised to follow arbitrary negotiation models. A process running on a collaboration framework associates members, a model of negotiation and a subject of collaboration together. The core operations defined in this layer are used by negotiation processes. An Encounter interface exposes the run-time state of a process involving a collection of participating members. The attributes of the Encounter interface are a subject of Encounter and a template that exposes the constraints. The result of the execution of an Encounter is either a success or a failure event. There are further interfaces supporting the specialisation of Encounter into Collaboration, Engagement and Voting definitions. The Collaboration type enables users to invoke transition operations that lead from initial to terminal states. A client joins an instance of Collaboration by establishing a Member role and interacts in the collaboration using operations like apply and invoke transitions and commands. The Collaboration interface has as attributes the active state which is an ordered sequence of state instances and a timeout list that exposes all active timeout transitions. On invocation of an apply operation a ‘transition’ is passed as argument which if successful will change the active state to the target and parent states in the transition.

Engagement interfaces enable the association of proof of engagement to an agreement by defining the security criteria to be applied during the engagement process. It handles open contracts or those requiring the explicit engagement of all participants and provides a persistent store for the registration of proofs. Voting interface has

the vote operation that takes one of the enumerated values *yes*, *no* or *abstain* as an input argument and determines the success or failure by counting the votes and deciding if the number of *yes* votes is greater than a percentage of the quorum. From the collaboration framework we have operations that define a group of users engaging in negotiation according to a subject and operations that manage the state of negotiation. As goal states are satisfied an engagement process can be launched for commitment and for setting up contracts between the agents. Participants need to comply to a protocol to ensure that they and all other participants following the same rule coordinate meaningfully. The protocol can be regarded the set of public rules that dictate the conduct of an agent towards other agents when carrying out some collaboration process.

B.18 Summary

The OMG has a number of specifications that allow objects to interoperate at a business level. Higher level frameworks can be built on CORBA to support electronic commerce applications. Some of the disadvantages of CORBA, such as lack of autonomy, can be solved by using an agent oriented approach. Agent architectures may provide solutions for sophisticated business transactions involving automated negotiation. The Negotiation Facility Framework consists of several layers of CORBA-compliant frameworks. Above the collaboration framework, negotiation processes can be thought as being abstract enough to come into the realm of agent oriented systems. These agent systems are at a natural level of abstraction over object oriented systems and inherit from them. Agent processes running over the negotiation facility interact, coordinate and thereby converge their goals towards agreements. Agent oriented applications are more suitable for supporting strategic decision making in unpredictable environments such as automated negotiation.

References

[1] Abadi M. and Lamport L. 1988. "The existence of refinement mappings". Theoretical Computer Science, 82:253--284, 1991.

[2] Abadi M., Alpern B., Apt K., Francez N., Katz S., Lamport L., Schneider F. 1991. "Preserving Liveness: Comments on Safety and Liveness from a Methodological Point of View". *Information Processing Letters* 40(3): 141-142 (1991)

[3] Aho A. V. and Ullman J. D. and Yannakakis M. 1979. "Modeling Communication Protocols by Automata". *Proc. 20th Annual IEEE Symposium on Foundations of Computer Science*, pp. 267-273, 1979.

[4] Aho A. V. and Ullman J. D. and Yannakakis M. 1982. "Bounds on the Size and Transmission Rate of Communication Protocols". *Computers and Math with Applications,* Vol. 8, 3, pp. 205-214, 1982.

[5] Allen, J. 1984. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123-154.

[6] Alogogianni, K. E. 2001. "Bilateral Agent Negotiation for Electronic Commerce". *MSc. In Advanced Computing Thesis*, *Imperial College.*

[7] Alpern B. and Schneider F. 1985. "Defining liveness". *Information Processing Letters*, 21:181-185.

[8] Amazon. 2002. http://amazon.co.uk

[9] Areces, C., Blackburn P. and Marx M. 1999. "A roadmap on the complexity of hybrid Logics." *Computer Science Logic*, *Proceedings of the 8th annual conference of the EACSL*. LNCS 1683, pp 307-321.

[10] Areces, C., Blackburn P. and Marx M. 1999b. "Hybrid Logics: Characterisation, interpolation and complexity." *Journal of Symbolic Logic*, 1999.

[11] Ashby, W.R. 1962. "Principles of the self-organizing system". *in H. Von Foerster and G.W. Zopf (eds.) Principles of Self-Organization*, Pergamon, p 255-278.

[12] Austin, J. L. 1962. "How to do things with words." *London Oxford University Press.*

[13] Bacchus, F. and Kabanza, F. 1996. "Using temporal logic to control search in a forward chaining planner". *In New Directions in AI Planning*, ISO Press. pp 141-153.

[14] Backhouse, R., Carre, B. 1975. "Regular algebra applied to path-finding problems". *J. Inst. Maths Applics. 1975, Volume 15, pp. 161-186.*

References

[15] Bailey, J.,Georgeff, M.,Kemp D., Kinny D. and Kotagiri R. 1995. "Active databases and agent systems - a comparison." *In T. Sellis, editor, Proceedings of the Second International Workshop on Rules in Database Systems,* Athens, Greece, LNCS 985, pages 342—356. Springer-Verlag

[16] Baldoni, M., Giordano, L., Martelli, A., and Patti V. 1998. "A Modal Programming Language for Representing Complex Actions". *Journal of Logic and Computation*.

[17] Barbuceanu, M. and Fox, M. S. 1995. "COOL: A Language for Describing Coordination in Multi-Agent Systems". *In Proceedings of the International Conference on Multi-Agent Systems*, San Francisco, CA.

[18] Bartlet, K. A., Scantlebury, R.A. and Wilkinson, P. T. 1969. "A Note on Reliable Full-Duplex Transmission over Half-Duplex Links", *Communications of the ACM,* Vol.12(5).

[19] Bauer, B. 2001b. "UML Class Diagrams: Revisited in the Context of Agent-Based Systems". *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 91-103.

[20] Bauer, B., Müller, J., Odell, J. 2001. "Agent UML: A Formalism for Specifying Multiagent Interaction". *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 91-103.

[21] Beam, C., Segev, A., Shanthikumar, J. G. 1996. " Electronic Negotiation through Internet-based Auctions". *Fisher Center for Information Technology and Management, University of California at Berkeley*, Working Paper 96-WP-1019.

[22] Ben-Ari, M., Halpern, J. Y. and Pnueli, A. 1982. "Deterministic Propositional Dynamic Logic". Finite Models, Complexity, and Completeness. *Journal of Computer and System Sciences*. 25(3): 402-417 (1982)

[23] Bench-Capon, T. J. M. 2001. "The Notion of an Ideal Audience in Legal Argument". *Artificial Intelligence and Law* 9(1): 59-71 (2001)

[24] Bensalem, S., Lakhnech Y., and Owre S. 1998. "Computing Abstractions of Infinite State Systems Compositionally and Automatically". In CAV'98. LNCS 1427, 1998.

[25] J. Billington, M. Diaz, G. Rozenberg. 1999. "Application of Petri Nets to Communication Networks". Lecture Notes in Computer Science, vol. 1605, Springer-Verlag, 1999,

[26] Bjorner, N., Browne, A., Colon, M., Finkbeiner, B., Manna, Z., Sipma, H. and Uribe T. 1999. "Verifying temporal properties of reactive systems: A step tutorial". *Formal Methods in System Design*.

[27] Blackburn, P. 2000. "Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto". *Proc. of the 1st. Method for Modalities Workshop. Amsterdam*. Special Issue of the Logic Journal of the IGPL. Vol 8:3, 339-625.

References

[28] Blackburn, P. and Siegleman, J. 1995. "Hybrid Languages." *Journal of Logic, language and Information*. 4(3) pp 251-272. Special issue on decompositions of first-order logic.

[29] Boehm, B W, Brown, JR., Kaspar, H., Lipow, M., MacLeod, G J. and Merritt, M J. 1978. "Characteristics of Software Quality". *New York, NY: North-Holland Publishing Company*, 1978.

[30] Boolos, G. S. and Jeffey, R. C. 1989. "Computability and Logic". *Cambridge University Press*. Third Edition. ISBN 0521389232.

[31] Boster, F. J., and Mongeau, P. 1984. "Fear-arousing persuasive messages". *In R. N. Bostrom, editor, Communication yearbook 8*. SAGE Publications, 1984

[32] Bradfield J and Stirling C. 1992. "Local model checking for inifinite state spaces". *Theoretical Computer Science,* 96:157--174, 1992.

[33] Bradshaw, J. 1995. "Software Agents". AAAI press/ The MIT press.

[34] Bradshaw, J. 1996. "KaoS: An open agent architecture supporting reuse, interoperability, and extensibility". *In tenth Knowledge Acquisition for knowledge-Based systems workshop*.

[35] Brazier, F., Van Eck, P. and Treur, J. 1997. "Modelling Competitive Cooperation of Agents in a Compositional Multi-Agent Framework". *International 13 Journal of Cooperative Information Systems*, 6:67-94.

[36] Brooks, R. 1991b. "Intelligence without representation", *AI, 47:139-159*.

[37] Brooks, R. 1990. "Elephants don't play chess." *Robotics and autonomous Systems*, 6:3-15.

[38] Brooks, R. 1991. "Intelligence without reason", *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91),* pages 569-595, Sydney, Australia.

[39] Burstall, R. M. 1974. "Program proving as hand simulation with a little induction." *In Information Processing 74*. Stockholm pp 308-312.

[40] Chang E., Manna Z., and Pnueli A. "The safety-progress classification". *In subseries F: Computer and System Science, NATO Advanced Science Institutes Series.* Springer-Verlag, 1992.

[41] Chavez, A. and Maes, P. 1996. "Kasbah: An agent marketplace for buying and selling goods". *In Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, April 1996.

[42] Chellas, B. 1980. "Modal Logic: An Introduction". *Cambridge University Press*, ISBN 0521295157.

[43] Cheng, D. T. and Covaci, S. 1997. "The OMG Mobile Agent Facility: A Submission". *Rothermel, K. and Popescu-Zeletin, R., Eds., Mobile Agents.* Springer-Verlag.

[44] Cherkasova, L. and Kotov, V. 1989." Descriptive and Analytical Process Algebras". Rozenberg, G.: Lecture Notes in Computer Science, Vol. 424; Advances in Petri Nets 1989, pages 77-104. Berlin, Germany: Springer-Verlag, 1990.

[45] Clark, k. l., Robinson P. and Hagen R. 1998. "Programming internet distributed DAI applications in Qu-Prolog". *In Multi-agent systems, (ed. C. Zhang and D. Lukose),* Springer-Verlag LNAI 1544

[46] Clarke E. and Emerson E. 1981. "Synthesis of synchronization skeletons for branching time temporal logic". In *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981* Lecture Notes in Computer Science, vol. 131, Springer-Verlag. 1981.

[47] Clearwater, S., Ed. 1995. "Market-Based Control: A Paradigm for distributed resource allocation." *World Scientific.*

[48] Cockburn, D. and Jennings, N. R. 1995. "ARCHON: A Distributed Artificial Intelligence System for Industrial Applications". *In: Foundations of Distributed Artificial Intelligence* (eds. G. M. P. O'Hare and N. R. Jennings), Wiley & Sons.

[49] Codish M. and Taboch C. 1999. "A semantic basis for the termination analysis of logic programs". *The Journal of Logic Programming*, 41:103-123, 1999.

[50] Cohen E. and Lamport L. 1998. "Reduction in tla". *In David Sangiorgi and Robert de Simone, editors, CONCUR'98 Concurrency Theory, volume 1466 of Lecture Notes in Computer Science*, pages 317--331. Springer-Verlag, 1998.

[51] Cohen, P. R. and Levesque, H. J. 1991. "Teamwork". *Nous*, 35 ,25(4):487-512.

[52] Cohen, M.D., March, J.G., and Olsen, J.P. 1972. "A garbage can model of organizational choice", *Administrative Science Quarterly*, 17, pp. 1-25.

[53] Cohen. P.R. and Levesque, H. J. 1990. "Intention is choice with commitment". *Artificial Intelligence* 42(3): 213-261.

[54] Conte, R. and Castelfranchi, C. 1995. "Cognitive and Social Action". *UCL Press, London.*

[55] Corkill, D. 1982. "A Framework for Organizational Self-Design in Distributed Problem Solving Networks" *PhD Dissertation*. *University of Massachusetts*.

[56] Cost, R., Chen Y., Finin, T., Labrou, Y. and Peng Y. 1999. "Modeling agent conversations with colored petri nets". *In Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 59-66.

[57] Cousot P. and Halbwachs N. 1978. "Automatic Discovery of Linear Restraints among Variables of a Program". *Conference Record 5th ACM Symp. on Principles of Programming Languages*, Tucson, 1978, pp. 84-96.

[58] Cox, B. 2001. "The Mainstreaming of E-commerce". *Ecommerce guide – news and trends*. 2001 INT Media Group.

References

[59] Cunningham, J. , Paurobally, S., Diacakis, A., McConnell, S., Gross, G., Lorenzen, L. 1998. "Requirements for Electronic Commerce". *In proc. Trends in Distributed Systems for Electronic Commerce*, Hamburg, Springer LNCS 1402.

[60] Cunningham, J. and Paurobally, S. 1999. "Negotiation Processes in Electronic Commerce" *AAAI Fall Symposium 1999. Workshop Modal & Temporal Logics based Planning for Open Networked Multimedia Systems*. Boston November 1999.

[61] DAML. http://www.daml.org

[62] Davis, E. 1990. "Representations of commonsense knowledge." *San Mateo*, CA: Morgan Kauffman.

[63] Decker, K. and Lesser, V. 1993. "Quantitave modeling of complex computational task environments." *Proc. Of the Eleventh National Conference on AI*, 217-224.

[64] Decker, K. and Lesser, V. 1997. "Designing a Family of Coordination Algorithms". *Readings in Agents.* Michael N. Huhns and Munindar P. Singh (eds.) p389-404. ISBN 1-55860-495-2

[65] Decker, K., Sycara, K. and Williamson, M. 1997. "Middle-agents for the Internet". *In Proc. of the 15th Joint Conf. on Artificial Intelligence, 1997.*

[66] Dederichs, F. Weber R. 1990. "Safety and Liveness From a Methodological Point of View". *Information Processing Letters* 36(1): 25-30.

[67] Dignum F. and Cortés U. 2001. "Agent-Mediated Electronic Commerce III". *LNCS 2003*, Springer-Verlag, 2001.

[68] Dignum F. 2001. "Agents, Markets, Institutions and Protocols". "Agent Mediated Electronic Commerce, The European Agentlink perspective", *LNCS-1991*, F. Dignum and C. Sierra (eds.). Springer-Verlag, pages 98-114.

[69] DOM, Document Object Model. http://www.w3.org/DOM/

[70] Dowell, M., Stephens, L., and Bonnell, R. 1995. "Using a domain knowledge ontology as a semantic gateway among databases". *In Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, International Joint Conference on Artificial Intelligence (IJCAI-95),* Montreal, Canada, August 1995.

[71] Durfee, E. 1998. "Planning in Distributed Artificial Intelligence." *In O'Hare and Jennings (eds.) Foundations of distributed artificial intelligence*, pp 31-247.

[72] Durfee, E. 1998. "Coordination of Distributed Problem Solvers." *Kluwer*, 1998.

[73] Durfee, E. 1999. "Distributed Problem Solving and Planning". *Mutiagent Systems A modern approach to distributed Artificial Intelligence.* Weiss Ed. MIT Press. ISBN 0262232030.

[74] Ebay. 2002. http://www.ebay.com.

[75] Elio R. and Haddadi, A. 1999. " On Abstract Models and Conversation Policies" *In Proc. Workshop on Specifying and Implementing Conversation Policies, Autonomous Agents'99 Conference*, Seattle, May, 1999.

[76] Emerson, E. A. 1990. "Temporal and modal logic". *Handbook of theoretical computer science*, vol. B. pp 995-1072.

[77] Fagin R. , Halpern, J. Y., Moses, Y. and Vardi, M. Y. 1995. "Reasoning about knowledge". *MIT Press*, Cambridge, Mass., 1995

[78] Fagin, R., Vardi, M. Y. 1985. "An Internal Semantics for Modal Logic: Preliminary Report". STOC 1985: 305-315

[79] Faratin, P. 2001. " Multi-Agent Contract Negotiation" *Socially Intelligent Agents - creating relationships with computers and robots "Multiagent Systems, Artificial Societies, and Simulated Organizations" Series* . Kluwer.

[80] Faratin, P., Klein, M., Samaya, H., Bar-Yam, Y. 2001. "Simple Negotiating Agents in Complex Games: Emergent Equilibria and Dominance of Strategies". *In Proceedings of the 8th Int Workshop on Agent Theories, Architectures and Languages (ATAL-01), Seattle, USA, pp. 42—53*.

[81] Faratin, P., Sierra C., and Jennings, N. R. 1998. "Negotiation Decision Functions for Autonomous Agents". *Int. Journal of Robotics and Autonomous Systems*, 24 (3-4).

[82] Faratin, P., Sierra C., Jennings, N. R. 2000. "Using Similarity Criteria to Make Negotiation Trade-Offs". *Proc. 4th International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 119-126.

[83] Faratin, P., Sierra C., Jennings, N. R. and Buckle, P. 1999. "Designing Responsive and Deliberative Automated Negotiators Negotiation". *Proc. AAAI Workshop on Negotiation: Settling Conflicts and Identifying Opportunities, Orlando, FL, 12-18.*

[84] Farquhar, A., Fikes, R. and Rice, J. 1996. "The Ontolingua server: A tool for collaborative ontology construction". *Technical report, Stanford KSL* 96-26.

[85] Fatima, S. S. , Wooldridge, M. and Jennings N. R. 2001. "Optimal negotiation strategies for agents with incomplete information". *Proc. 8th Int. Workshop on Agent Theories, Architectures and Languages* (ATAL), Seattle, USA, 53-68.

[86] Feldbrugge, F. and Jensen, K. 1987. "Petri Nets: Applications and Relationships to Other Models of Concurrency". *In Lecture Notes in Computer Science, Vol. 255, pages 20-61.*

[87] Ferguson, I. 1992. "Touring Machines: Autonomous Agents with Attitudes". *IEEE Computer 25*(5): 51-55

[88] Fich F.E.. 1998. "End-to-end Communication", *Proceedings of the 2nd Int. Conf. on Principles of Distributed Systems*. 1998.

[89] Fikes, R.E. and Nilsson, N.J. 1971. "STRIPS: A new approach to the application of theorem proving to problem solving". *Artificial Intelligence* 2(3-4).

[90] Finin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzson, R., McKay, D., McGuire, J., Pelavin, R., Shapiro, S., Buffalo, S., Beck, C. 1993. "Specification of KQML, Agent Communication Language" *The DARPA Knowledge Sharing Initiative External Interfaces Working Group, February, 1993*.

[91] FIPA .2001e. "FIPA Iterated Contract Net Interaction Protocol". http://www.fipa.org/specs/fipa00030/XC00030F.html

[92] FIPA. 2001a. "FIPA Interaction Protocol Library specification". http://www.fipa.org/specs/fipa00025/XC00025E.html

[93] FIPA. 2001b. "FIPA Request Interaction Protocol". http://www.fipa.org/specs/fipa00026/XC00026F.html

[94] FIPA. 2001c. "FIPA Request When Interaction Protocol". http://www.fipa.org/specs/fipa00028/XC00028F.html

[95] FIPA. 2001d. "FIPA Contract Net Interaction Protocol". http://www.fipa.org/specs/fipa00029/XC00029F.html

[96] FIPA. 2001f. "FIPA Brokering Interaction Protocol". http://www.fipa.org/specs/fipa00033/XC00033F.html

[97] FIPA. 2001g. "FIPARecruiting Interaction Protocol". http://www.fipa.org/specs/fipa00034/XC00034F.html

[98] FIPA. 2001h. "FIPA English Auction Interaction Protocol". http://www.fipa.org/specs/fipa00031/XC00031F.html

[99] FIPA. 2001i. "FIPA Dutch Auction Interaction Protocol". http://www.fipa.org/specs/fipa00032/XC00032F.html

[100] FIPA.. 1997. Agent Communication Language Technical report. Foundation for Intelligent Physical Agents. http://www.fipa.org

[101] FIPA.. 2001. Foundation for Intelligent Physical Agents. http://www.fipa.org

[102] Fischer, M. J. and Ladner, R.E. 1979. "Propositional dynamic logic of regular programs." *Journal of Computational System Science*. Vol 18, pp 194-211.

[103] Foldoc. "Free on-line dictionary of computing". http://foldoc.doc.ic.ac.uk

[104] Francez, N., and Pnueli, A. 1978. "A proof method for cyclic programs", *Acta Inf.,* vol 9, 1978, 138-158

[105] Franklin, S. and Graesser, A. 1996. "Is it an agent, or just a program?: A taxonomy for autonomous agents". *In Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag,

[106] Galbraith, J. 1973. "Designing Complex Organizations" *Addison-Wesley*

[107] Ganzinger H. and Waldmann U. 1993. "Termination proofs of well-moded logic programs via conditional rewrite systems". In *Proceedings of the 3rd International*

Workshop on Conditional Term Rewriting Systems, volume 656 of *Lecture Notes in Computer Science*, pages 113-127, Berlin, 1993. Springer-Verlag.

[108] Gasser L. and A. Majchrzak. 1992. "HITOP-A: A Tool To Facilitate Interdisciplinary Manufacturing System Design" *International Journal of Human Factors in Manufacturing*, 2(3), pages 255--276, 1992.

[109] Gasser L., Braganza C., and Herman N.1987. "Implementing distributed artificial intelligence systems using mace". *In Proceedings of the Third IEEE Conference on Artificial Intelligence Applications*, pages 315-320.

[110] Gasser, L. 1998. "Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics". *Readings in Agents.* Michael N. Huhns and Munindar P. Singh (eds.) p389-404. ISBN 1-55860-495-2

[111] General Magic. 1995. "Telescript Language Reference". *General Magic, Inc. 420 North Mary Avenue, Sunnyvale, CA 94086.* October 1995. http://www.generalmagic.com/

[112] Gerbrandy, J. and Groeneveld, W. 1997. "Reasoning about Information Change". *Journal of logic, language and information*, 6:147-169, 1997.

[113] German S M. and Wegbreit B. 1975. "A synthesizer of inductive assertions". *IEEE Transactions on Software Engineering*, 1(1):68--75, March 1975.

[114] Giacomo, G. and Lenzerini, M. 1995. "PDL-based framework for reasoning about actions". *In M. Gori and G. Soda (Eds.), Topics in Articial Intelligence*. LNAI 992.

[115] Giddens, A. 1984. "The constitution of society". *University of California Press.*

[116] Giordano, L., Martelli, A. and C. Schwind. 1998. "Dealing with concurrent actions in modal action logics". *In H. Prade, editor, ECAI'98*. John Wiley & Sons. 537-541.

[117] Goldblatt, R. 1992. Logics of Time and Computation, *Lecture Notes, Center for the Study of Language and Information 1987.*

[118] Gordon M. and Melham T. 1993. "Introduction to HOL: A Theorem Proving Environment for Higher-Oder Logic". *University Press*, Cambridge, 1993.

[119] Graf S. and Sifakis J. 1987. "Readiness semantics for regular processes with silent actions" *Proc. ICALP 87 (Th.Ottman, ed.) Karlsruhe, LNCS 267, Springer-Verlag*, pp.115-125.

[120] Gray, R. 1996. "Agent Tcl: A flexible and secure mobile agent system" *In Proc. of the Fourth Annual Tcl/Tk Workshop*, pages 9-23, Monterey, Cal., July 1996.

[121] Grosz, B. and Kraus, S. 1996. "Collaborative plans for complex group actions." *Aritificial Intelligence*, 86:269-358.

[122] Guardian Newspaper, Friday 3rd August 2001.

[123] Guilfoyle, C.,Jeffcoate, J. and Stark, H. 1997. "Agents on the Web: Catalyst for E-Commerce". Ovum Ltd, London, Apr. 1997.

[124] Guttman, R., Mouskas, A. and Maes, P. 1998. "Agent-Mediated electronic commerce: A survey". *Knowledge Engineering Review,* 13(2): 147-159.

[125] Hailpern B. T. and Owicki S. S. 1980. "Verifying Network Protocols using Temporal Logic". *Technical Report 192*. Computer Systems Laboratory Stanford University.

[126] Halpern and Moses. 1990. Knowledge and common knowledge in a distributed environment. Journal of the ACM, 37:549-587.

[127] Halpern J. Y. and Shoham Y. 1991. "A propositional modal logic of time intervals". *Journal of the ACM*, 38(4):935-962, 1991.

[128] Halpern, J.Y. and Zuck, L.D. 1992. **"**A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols". *J. of the ACM*, 39(3).

[129] Hanks and McDermott 1986. "Default reasoning, nonmonotonic logic, and the frame problem." *Proceedings* of *AAAI 1986*, 328-333.

[130] Haar, S., Kaiser L., Simonot-Lion, F. and Oussaint, J. 2000. "On Equivalence between Timed State Machines and Time Petri Nets". INRIA Research report RR-4049.

[131] Harel, D. 1984., "First order dynamic logic". *Extensions of Classical Logic, Handbook of Philosophical Logic II*. pp.497-604.

[132] Harel, D. and Naamad, A. 1996. "The Statemate Semantics of Statecharts". *ACM Transactions Software Engineering Method 5:4 October 1996.*

[133] Harel, D., Naamad, A.,Lachover, H.,.Pnueli, A., Politi, M. 1990. " Statemate: A Working Environment for the Development of Complex Reactive Systems". *IEEE Transactions of Software Engineering Vol16, No4, April 1990.*

[134] Harel, D., Politi, M. 1998. "Modeling reactive systems with statecharts, The Statemate approach". *McGraw-Hill*.

[135] Harel, D., Rosner, R., and Vardi, M. 1990a. "On the power of bounded concurrency III: Reasoning about programs". *Proceedings of Fifth Annual IEEE Symposium on Logic in Computer Science*, 478-488, Philadelphia, Pennsylvania.

[136] Hennessy M. and Milner R. 1985 "Algebraic laws for nondeterminism and concurrency". Journal of the ACM (JACM). 32(1), 137—161.

[137] Hintikka, J. 1969. "Semantics for propositional attitudes," *In Models for Modalities,* Reidel: Dordrecht, the Netherlands.

[138] Hoare C. A. 1980. "Communicating sequential processes, On the construction of programs" *R.McKeag & A Macnaghten, eds. Cambridge University press,* pp. 229-254.

[139] Holzmann G. J. 1997. "The model checker SPIN". *IEEE Trans. on Software Engineering, 23(5):279--295*, 1997.

[140] Houston, P. J., Wilkie, F. G., Anderson, T. J. 1998. "Component-based development, CORBA and RM-ODP", *IEE Proceeding-Software*, 145(1):22-28.

[141] Howard, R.A. and Matheson, J.E. 1981. "Influence diagrams." *In the principles and applications of decision analysis*. pp 720-762.

[142] Huhns, M. N. and Singh M. P (eds.) 1997. "Readings in Agents". p371-379. ISBN 1-55860-495-2

[143] Huhns, M. N. and Stephens, L. 1999. "Multiagent Systems and Societies of Agents". *In Multiagent Systems A modern approach to distributed Artificial Intelligence*. MIT Press. ISBN 0262232030.

[144] Huhns, M. N. and Weiss, G. (eds.). 1998. "Special Issue on Multiagent Learning". *Machine Learning Journal*. Vol 33(2-3).

[145] Huhns, M., and Singh, M. 1997. "Ontologies for Agents". *In IEEE Internet Computing* November-December 1997, pp. 81-83

[146] Hunsberger, L. and. Grosz, B. J. 2000. "A combinatorial auction for collaborative planning". *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000),* pages 151-158, 2000.

[147] Jackson, M. 1975. " Structured Design". *Academic Press 1975*

[148] Jancar, P. 1995. "High Undecidability of Weak Bisimilarity for Petri Nets." *In: Mosses, P.D.; Nielsen, M.; Schwartzbach, M.I.: Lecture Notes in Computer Science, Vol. 915; TAPSOFT'95: Theory and Practice of Software Development, Aarhus, Denmark*, May 22-26, 1995, pages 349-363. Springer-Verlag, 1995.

[149] Jaynes, E. 1985. "Bayesian methods: General background". *In Maximum Entropy and Bayesian methods*. Cambridge University Press, pp 1-25.

[150] Jennings, N. 1995. "Controlling cooperative problem solving in industrial multi-agent systems using joint intentions". *Artificial Intelligence*, 75.

[151] Jennings, N. and Wooldridge, M. 1997. (eds). "Agent Technology". *Springer.*

[152] Jennings, N. and Wooldridge, M. 1996. "Software Agents". *IEE Review* 42(1).

[153] Jennings, N. R., Faratin, P., Lomuscio, A., Parsons, S., Sierra, C. and Wooldridge, M. 2001. "Automated negotiation: prospects, methods and challenges" *Int. J. of Group Decision and Negotiation 10* (2) 199-215.

[154] Jennings, N., Faratin, P.,Johnson, M., O'Brien, P. and Wiegand, M. 1996. "Using Intelligent Agents to Manage Business Processes", *Proc. First Int. Conf. on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM96).*

[155] Jennings, N., Norman, T. and Faratin, P. 1998. "ADEPT: An agent-based apporach to business process management". *ACM ISGMOD Record*, 27(4): 32-39.

[156] Jennings, N., Parsons, S., Sierra, C. and Faratin,P. 2000. "Automated Negotiation". *Proc. 5th Int. Conf. on the Practical Application of Intelligent Agents and Multi- Agent Systems (PAAM-2000*), Manchester, UK, 23-30.

[157] Jennings, N., Sycara, K.and Wooldridge, M. 1998. "A roadmap of agent research and development". *International Journal of Autonomous Agents and Multi-Agent Systems.*

[158] Jensen, K. 1993. "An Introduction to the Theoretical aspects of Coloured Petri Nets". *Lecture Notes in Computer Science, Vol. 803; A Decade of Concurrency, pages 230-272*. Springer-Verlag, June 1993.

[159] Jensen, K. 1994. "Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use" *EATCS Monographs on Theoretical Computer Science. Berlin: Springer-Verlag*.

[160] Jensen, K. 1994. "Condensed State Spaces for Symmetrical Coloured Petri Nets." *Formal Methods in System Design Vol. 9, pages 7-40. Kluwer Academic Publishers*..

[161] Jensen, K. 1998. "An Introduction to the Practical Use of Coloured Petri Nets". *Lecture Notes in Computer Science, Vol. 1492: Lectures on Petri Nets II: Applications. Springer-Verlag, 1998.*

[162] In: Reisig, W.; Rozenberg, G.: Lecture Notes in Computer Science, Vol. 1492: Lectures on Petri Nets II: Applications. Springer-Verlag, 1998. ISBN: 3-540-65307-4.

[163] Jin, Y and Levitt, R. 1996. "The Virtual Design Team: A Computational Model of Project Organizations", *Computational and Mathematical Organisation Theory.* 2(3).

[164] Jones, A. and Firozabadi B. 2000. "On the Characterisation of a trusting agent: Aspects of a formal approach". *In C. Castelfranchi and Y. Tan (eds): Trust and Deception in Virtual Societies*. Kluwer Academic Press.

[165] Kaelbling, L. P., Littman, M. L. and Moore, A. W. 1996. "Reinforcement learning: A survey". *Journal of Artificial Intelligence Research.* 4:237--285

[166] Kakas A., Kowalski R., Toni F. 1992. "Abductive Logic Programming". *Journal of Logic and Computation* 2(6): 719-770.

[167] Kakas, A. and Miller, R. 1997. "A simple declarative language for describing narratives with actions." *The journal of Logic Programming*. Vol 31(1-3), pp. 157-200.

[168] Keller R. 1976. "Formal Verification of Parallel Programs". *Communications of the ACM*, vol. 7, 1976

[169] Keller, R. 1976. "Formal Verification of Parallel Programs". *Communications of the ACM*, vol. 7. 1976.

[170] Kindler, E. 1994. "Safety and Liveness Properties: A Survey". *In EATCS* [EAT94], pages 268+.

[171] Knuth, D. E. and Moore, R. W. 1975. "An analysis of alpha-beta pruning". *Artificial Intelligence*, 6(4):293-326.

[172] Koning J L and Huget M P, Jun Wei, and Xu Wang. 2001. "Extended Modeling Languages for Interaction Protocol Design". *Proc. of Agent-Oriented Software Engineering (AOSE) 2001, Agents 2001, Montreal*. p. 93-100.

[173] Koning J L and Huget M P. 2001. "A Semi-Formal Specification Language Dedicated to Interaction Protocols", *H. Kangassalo and H. Jaakkola and E. Kawaguchi (eds.), Information Modeling and Knowledge Bases XII*, Frontiers in Artificial Intelligence and Applications, IOS Press, 2001.

[174] Konolige, K. 1986. "A Deduction Model of Belief". *Pitman Publishing: London and Morgan Kaufmann*: San Mateo, CA

[175] Korf, R. E. 1988. "Search in AI: A survey of recent results". *In H.Shrobe, editor, Exploring Artificial Intelligence*. Morgan-Kauffman.

[176] Kosaraju, S. R. 1982. "Decidability of reachability in vector addition systems," *14th Annual ACM Symp. Theory and Computing*. P 267-281.

[177] Kowalski R. and Sergot, M. 1986. "A logic-based calculus of events". *New generation computing* 4(1):67-95.

[178] Kozen, D. 1983. "Results on the propostional mu-calculus". *Theoretical Computer Science 27, 333-354*.

[179] Kraus, S. 1996. "An Overview of Incentive Contracting". *AI journal 83(2),* 297-346.

[180] Kraus, S., Sycara, K. and Evenchik, A. 1998. "Reaching Agreements through Argumentation: a Logical Model and Implementation". *Artificial Intelligence* 104:1--70.

[181] Kraus. S. 2001. "Automated Negotiation and Decision Making in Multi-agent Environments." *In Proceedings of 9th ECCAI Advanced Course and EASSS 2001*. Springer, LNAI 2086.

[182] Kraus. S. 2001a. " Strategic Negotiation in Multi-Agent Environments". *MIT Press,* Cambridge, USA.

[183] Kripke, S. A. 1963. "Semantical analysis of modal logic I: Normal propositional calculi," *Zeitschrift fur mathematics Logik und Grundlagen der Mathematik*, pp 67-96.

[184] Krulwich, B. 1996. "The bargainfinder agent: Comparison price shopping on the internet". In J. Williams, ed., Bots and Other Internet Beasties. SAMS.NET, 1996. http://bf.cstar.ac.com/bf/.

[185] Kuwabara, K., Ishida, T. and Osata, N. 1995. "AgenTalk: Describing Multiagent Coordination Protocols with Inheritance". *In proceedings of the 7th IEEE International Conference on Tools for Artificial Intelligence*, pages 460-465.

[186] Labrou, Y. 2001. "Standardising Agent Communication". *In Proceedings of 9th ECCAI Advanced Course and EASSS 2001*. Springer, LNAI 2086.

[187] Labrou, Y. and Finin, T. 1997. "Semantics for an agent communication language". *The fourth International workshop on Agent Theories, Architectures and Languages*. Rhode Island, USA.

[188] Labrou, Y. and Finin, T. 1997b. **"**Semantics and conversations for an agent communication language". *In Proceedings of the Fifteenth International Joint*

Conference on Artificial Intelligence (IJCAI-97), pages 584--591, Nagoya, Japan. *Communications of the ACM*, 21:666--677.

[189] Lamport, L. 1977. "Proving the correctness of multiprocess programs". *IEEE Transactions Software Eng, Se-3*, 2, (March 1997), 125-143.

[190] Lamport, L. 1980. "The "Hoare Logic" of concurrent programs". *Acta Inf 14*, *1*. 21-37

[191] Lamport, L. 1980a. "Sometimes is sometimes "not never" - on the temporal logic of programs". *In Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, pages 174--185, January 1980.

[192] Lamport, L. 1994. "A temporal logic of actions". *ACM Transactions on Programming Languages and Systems*, 16(3):872-- 923, March 1994.

[193] Last-Minute. 2002. http://www.last-minute.com

[194] Lehmann, D., O'Callaghan, L.I. and Shoham, Y. 1999. "Truth revalation in rapid, approximately efficient combinatorial auctions". *In ACM Conference on Electronic Commerce*, 1999.

[195] Lendaris,G. 1964. "On the definition of self-organizing systems." *IEEE Proceedings 52*. p324-325.

[196] Lichtenstein, O., Pnueli, A., Zuck, L. 1985. "The glory of the past". *Proc. Workshop on Logics of Programs*, Brooklyn, 1985, Springer-Verlag, LNCS 193, pp. 196--218.

[197] Lieberman, H. 1997. "Autonomous interface agents". *In Proc. ACM Conference on Computers and Human Interface*, Atlanta, GA, 1997.

[198] Lin, F. and Shoham, Y. 1992. "Concurrent actions in the situation calculus." *In Proc. AAAI-92*, pp. 590-595.

[199] Lin, F., Norrie, D H, Shen, W and Kremer R. 1999. "Schema-based approach to specifying conversation policies". *In Working Notes of the Workshop on Specifying and Implementing Conversation Policies, Third International Conference on Autonomous Agents*, pages 71--78.

[200] Lomuscio, A., Wooldridge, M. and Jennings, N. 2000. "A Classification Scheme for Negotiation in Electronic Commerce. *In Agent Mediated Electronic Commerce. The European AgentLink Perspective. C. Sierra and F. Dignum, editors.* ISBN: 3-540-41671-4. Springer Verlag. pages 61-77.

[201] Loredo, T. 1990. "From Laplace to supernova SN1987A: Bayesian inference in astrophysics". *In Maximum Entropy and Bayesian methods*. Kluwer Academic Press, pp 81-142.

[202] Lufthansa. http://www.lufthansa.com

[203] Maes, P. 1991. "Designing Autonomous Agents". Cambridge MIT Press.

[204] Maes, P., "Intelligent Software." *In: Labile Ordnungen, Hans-Bedrow-Institut*, 1997.

[205] Maes, P., Guttman, R., Moukas, A. 1999. ”Agents that buy and sell”. *Communications of the ACM 42*, 3 (Mar. 1999), p. 81

[206] Malone., T. W. 1987. “Modeling Coordination in Organizations and Markets”. *Management Science,* 33(10):1317--1332, 1987.

[207] Manna Z. and Pnueli A. 1995. “Temporal verification of reactive systems – safety”. *Springer-Verlag ISBN 0387944591*.

[208] Manna, Z. and Sipma H. 1998. “Deductive Verification of Hybrid Systems using SteP”. *In Hybrid systems: computation and control*. *International Workshop* LNCS 1386, Springer-Verlag.

[209] Manning, P. 1977. “Rules in an Organizational Context.” *In Organizational Analysis, Sage.*

[210] Marsh S. P. 1994. “Formalising Trust as a Computational Concept”. *Ph.D. thesis, University of Stirling*, *Dept. of Comp. Science and Mathematics*. April 1994.

[211] Martin, D., Cheyer, A., and Moran, D. 1999b. “The Open Agent Architecture: a framework for building distributed software systems." *Applied AI* , 13(1/2):91-128.

[212] Martin, F., Plaza, E. and Rodr ' iguez-Aguilar, J. 1999. “Conversation protocols: Modeling and implementing conversations in agent-based systems”. *In Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, *Third International Conference on Autonomous Agents* ,pages 49-58.

[213] Matos, N., Sierra, C., and Jennings, N. 1998. "Determining Successful Negotiation Strategies: An Evolutionary Approach*". Proceedings of the Third International Conference on Multi-Agent Systems* ICMAS'98, Paris, 1998, pp. 182-189.

[214] Mayfield J., Labrou Y. and Finin T. “Evaluation of KQML as an agent communication language”. *In M. Wooldridge, J. P. Mueller, and M. Tambe, editors, Intelligent Agents II: Agent Theories, Architectures, and Language*. Springer-Verlag Lecture Notes in AI, Volume 1037, 1996.

[215] Mayr E. W. 1984. “An algorithm for the general Petri net reachability problem”. *SIAM J. Comp.* 13(3) 441-460.

[216] McBurney, P., Parsons, S. and Johnson M. W. 2002. “When are two protocols the same?” In Agent Communication Languages and Conversation Policies, Proceedings of an AAMAS-02 Workshop, Bologna, Italy.

[217] McCarthy J. and Costello T. 1998. “Combining narratives”. In *A.G. Cohn and L.K. Schubert, editors, Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98),* pages 48-59.

[218] McCarthy, J. 1980. “Circumscription – a form of non-monotonic reasoning.” *Artificial Intelligence*, 13, 27-39.

[219] McCarthy, J. 1986. "Applications of circumscription to formalizing common-sense knowledge". *Artificial Intelligence*, 28:89-116.

[220] McCarthy, J. and Hayes, P. 1969. "Some Philosophical Problems from the Standpoint of Artificial Intelligence" *In Machine Intelligence 4*. pp 463-502

[221] Meyer, J.-J. C. and van der Hoek, W. 1995. "Epistemic Logic for Artificial Intelligence and Computer Science" *volume 41 of Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press: Cambridge, England, 1995.

[222] Miller, J. A., Sheth, A. P., Kochut, K. J. ; Wang, X. 1996. "CORBA-Based Run-Time Architectures for Workflow Management Systems". *Journal of Database Management, Special Issue on Multidatabases*, 7(1) (1996), pp. 16-27.

[223] Milner R. 1989. "Communication and Concurrency". *Prentice Hall*, 1989.

[224] Minar, N., Burkhart, R., Langton, C., Askenazi, M. 1996. "The Swarm Simulation System, A Toolkit for Building Multi-Agent Simulations". *http://www.santafe.edu/projects/swarm/overview/overview.html.*

[225] Mitchell, T. 1997. "Machine learning". *McGraw-Hill*.

[226] Moore, S. 1999. "On conversation policies and the need for exceptions". *In Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, *Third International Conference on Autonomous Agents.* pages 19-28.

[227] Moss S. and Edmonds B.1997. "A formal preference-state model with qualitative market judgements". *The International Journal of Management Science*, 25(2):155-169

[228] Muggleton S. 1999. "Inductive Logic Programming: Issues, Results and the Challenge of Learning Language in Logic". *Artificial Intelligence* 114(1-2): 283-296.

[229] Müller, J 1996. "The Design of Intelligent Agents - A Layered Approach". *Springer.*

[230] Murata, T. 1989. "Petri Nets: Properties, Analysis and Applications". *Proceedings of the IEE*, Vol 77(4). April 1989.

[231] Naur, P. (ed.) 1960. "Revised Report on the Algorithmic Language ALGOL 60." *Communications of the ACM,* Vol 3 (5), pp. 299-314.

[232] Neiger, G. and Tuttle M. R. 1993. "Common knowledge and consistent simultaneous coordination". *Distributed Computing,* 6(3):181-192.

[233] Nodine M. and Unruh A. 1997. "Facilitating open communication in agent systems: the InfoSleuth infrastructure". *In Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages*, 1997. MCC-INSL-113-98.

[234] Noriega, P. and Sierra C. 1996. "Towards layered dialogical agents". *In Proceedings of the ECAI'96 Workshop Agents Theories, Architectures and Languages ATAL'9*.

[235] Noriega, P. and Sierra C. editors. 1999. "Agent Mediated Electronic Commerce" *(LNAI Volume 1571).* Springer-Verlag: Berlin, Germany.

[236] Nouvelles-frontieres. 2002. http://www3.nouvelles-frontieres.fr/nf

[237] Nowostawski, M., Purvis, M., and Cranefield, S., "A Layered Approach for Modelling Agent Conversations ", *Proceedings of the 2nd International Workshop on Infrastructure for Agents, MAS, and Scalable, MAS, 5th International Conference on Autonomous Agents (2001)* 163170.

[238] Nwana, H. S. 1996. "Software agents: An Overview". *Intelligent Systems Research, BT Laboratories, Knowledge Engineering Review.* Vol. 11, No. 3, pp 205-244.

[239] Odell, J., Parunak, H.V.D., Bauer B. 2000. "Extending UML for Agents". *Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, Gerd Wagner, Yves Lesperance, and Eric Yu eds., Austin, TX.

[240] Odell, J., Parunak, H.V.D., Bauer B. 2001. "Representing Agent Interaction Protocols in UML". *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 121–140.

[241] Ohlebusch, E. 1999. "Transforming conditional rewrite systems with extra variables into unconditional systems". In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 111-130, Berlin, 1999. Springer-Verlag.

[242] OMG, Object Management Group. 1998a. "Combined Business Object Facility Interoperability Specification", *http://www.omg.org/pub/docs/bom/98-05-03*

[243] OMG, Object Management Group. 1998b. "Mobile Agent Facility Specification", http://www.omg.org/pub/docs/cf/02-06-97

[244] OMG, Object Management Group. 1999. Task and Session Faciltiy. http://www.omg.org

[245] OMG, Object Management Group. 2000. "The Common Object Request Broker: Architecture and Specification, 2.4". http://www.omg.org.

[246] Orfali, R. and Harkey, D. 1997a. "Client-server programming with Java and CORBA", *John Wiley & Sons.*

[247] Orfali, R., Harkey, D., Edwards, J. 1997b. "Instant CORBA". *John Wiley & Sons.*

[248] OSM Consortium. 1997. "WP1 Task 1.6, Deliverable 6 Consolidated Analysis". *http://www.osm.net*

[249] OSM Consortium. 1997. "WP2, OSM Reference Architecture". *http://www.osm.net*

[250] OSM SARL. 1998. Negotiation facility. , http://www.omg.org, EC/99-03-06

[251] Owicki S. and Lamport L. 1982. "Proving Liveness Properties of Concurrent Programs." *ACM Transactions on Programming Languages and Systems*. Vol. 4, No. 3. July 1982, pages 455 – 495.

[252] Özsu M. T. and Valduriez P. 1999. "Principles of Distributed Database Systems", *Second Edition Prentice Hall ISBN 0-13-659707-6.*

[253] Panconesi A. 1997. http://www.nada.kth.se/kurser/kth/2D5340/wwwbook/node20.html

[254] Parsons, S., Sierra, C. and Jennings, N. R. 1998. "Agents that reason and negotiate by arguing". *Journal of Logic and computation*, 8(3):261-292

[255] Parunak Van D. H.1996. "Visualizing agent conversations: Using enhanced Dooley graphs for agent design and analysis". *In Proc. of the 2nd International Conference on Multi-Agent Systems* (ICMAS '96).

[256] Passay, S. and Tinchev, T. 1991. "An Essay in Combinatory Dynamic Logic". *Information and Computation* 93(2): 263-332 (1991)

[257] Paurobally, S. and Cunningham, J. 2000a. "Specifying the Processes and States of Negotiation". *In Agent Mediated Electronic Commerce. The European AgentLink Perspective*. C. Sierra and F. Dignum, editors. ISBN: 3-540-41671-4. Springer Verlag. pages 61-77.

[258] Paurobally, S. and Cunningham, J. 2000b. "Processes and States of Negotiation and Underlying Architecture." *In Practical Reasoning Agent Workshop.* FAPR 2000. London. ISSN 1469-4166.

[259] Paurobally, S. and Cunningham, J. 2002a. "Verification of Protocols for negotiation between agents." *Proceedings of European Conference on AI 2002. (ECAI 2002), Lyon France,* July 2002.

[260] Paurobally, S. and Cunningham, J. 2002b. Safety and Liveness of Negotiation Protocols. *Artificial intelligence and the simulation of behaviour Convention. (AISB2002). Intelligent Agents in virtual markets track*. London April 2002.

[261] Pearl, J. 1984. "Heuristics: Intelligent Search Strategies for Computer Problem Solving". Addison-Wesley

[262] Pearl, J. 1988. "Probabilistic reasoning in intelligent systems: Networks of plausible inference". *Moragn Kaufmann*, San Mateo, CA.

[263] Pentland, B.T. 1995. "Grammatical models of organizational processes". *Organization Science* 6, 5, pp.541-556. 1995.

[264] Petri, C.A. 1966. "Communication with Automata". Vol. 1. Applied Data Research, Princeton, AF 30(602)-3324, 1966

[265] Pitt, J., and Mamdani, A. 1999. "Communication protocols in multi-agent systems". *In Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, *Third International Conference on Autonomous Agents* pages 39-48.

[266] Pitt, J., Anderton M., Cunningham, J. 1996. "Normalized Interactions between Autonomous Agents, A Case Study in Inter-Organisational Project Management", *CSCW 5, pp 210-222*.

References

[267] Pnueli A. 1977. "The temporal semantics of concurrent programs." *Theoretical Computer Science* 13:1-20.

[268] Pnueli A. 1979. "The temporal logic of programs". *In proceedings of the 18th Symposium on the Foundation of Computer Science, IEEE, Providence,* pp 46-57.

[269] Pnueli A. 1985. "Linear and branching structures in the semantics and logics of reactive systems", *proc. 12th ICALP (W.Brauer, ed.) Nafplion, Greece*, LNCS 194, Springer-Verlag, pp. 15-32.

[270] Poole D., Mackworth A., Goebel R. 1998. "Computational Intelligence. A logical approach." *Oxford University Press ISBN* 0195102703.

[271] Poslad, S. and Calisti, M. 2000. "Towards improved trust and security in FIPA agent platforms". *Autonomous Agents 2000 Workshop on Deception, fraud, and trust in agent societies*, Barcelona, June 2000.

[272] Poslad, S. and Charlton, P. 2001. "Standardising Agent Interoperability: The FIPA approach." *In Proceedings of 9th ECCAI Advanced Course and EASSS 2001*. Springer, LNAI 2086.

[273] Prakken H., Sergot M.J. 1997. "Dyadic Deontic Logic and Contrary-to-duty Obligations". *Defeasible Deontic Logic: Essays in Nonmonotonic Normative Reasoning*. D. Nute (ed.), Synthese Library No. 263, Kluwer Academic Publishers, pp 223-262.

[274] Pratt, V. R. 1976. "Semantical considerations on Floyd-Hoare logic". *Proceedings of 17th IEEE Symposium, Foundations of Computer Science, pp. 109-121.*

[275] Pratt, V. R. 1981. "A decidable mu-calculus: Preliminary report." *Proceedings 22nd Annual Symposium on Foundations of Computer Science*. pp. 421-427.

[276] Pratt, V.R.1976. "Semantical considerations on Floyd-Hoare logic". *In Proc. 17th Ann. IEEE Symp. on Foundations of Comp. Sci*., pages 109--121.

[277] Purvis, M. K., Cranefield, S. J. S., Nowostawski, M., and Purvis, M. A. 2002. "Multi-Agent System Interaction Protocols in a Dynamically Changing Environment", *Information Science Discussion Paper Series, Number 2002/04*.

[278] QXL. 2002. http://www.qxl.com

[279] Raiffa, H. 1982. *The Art and Science of Negotiation*. Harvard University Press, 1982

[280] Rao A. S. and Georgeff, M. P. 1991. "Modeling Rational Agents within a BDI-Architecture*", In R. Fikes and E. Sandewall, editors, International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Massachusetts.

[281] Rao A. S. and Georgeff, M. P. 1992. "An Abstract Architecture for Rational Agents", *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, Boston 1992.*

[282] Rao A. S. and Georgeff, M. P. 1995. "BDI agents: From theory to practice". *In V. Lesser (ed.): Proc. of the First Int. Conf. on Multi-Agent Systems (ICMAS-95).* 312-319. MIT Press, 1995.

[283] Rasmusen, E. 1989. "Games and Information: An Introduction to Game Theory". *Basil Blackwell, Oxford, U.K. and Cambridge*, *Mass.*

[284] Reisig, W. 1985. "Petri Nets, An Introduction". EATCS, Monographs on Theoretical Computer Science, W.Brauer, G. Rozenberg, A. Salomaa (Eds.), Springer Verlag, Berlin, 1985.

[285] Reiter, R. 1991. "The frame problem in the situation calculus: A simple solution (sometiems) and a completeness result for goal regression." *In Artificial Intelligence and mathematical theory of computation*: *Papers in honor of John McCarthy*, 359-380.

[286] Rhodes, B. 2000. "Margin notes: building a contextually aware associative memory". *Intelligent User Interfaces 2000*: 219-224.

[287] Rich, C. and Sidner, C. 1997. "COLLAGEN: When agents collaborate with people." *In Proceedings of the International Conference on Autonomous Agents (Agents'97*).

[288] Richters M. and Gogolla M. 1998. "On formalizing the UML object constraint language OCL". *In Proceedings, Conceptual Modeling, LNCS 1507*, pages 449--464. Springer, 1998.

[289] Rivest, R., Shamir A. and Adlemann, L. 1978. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." *Communications of the ACM*, 21: 120-126. February 1978.

[290] Robles, S., Poslad, S., Borrell, J., Bigham, J. 2001. "Adding security and privacy to agents acting in a marketplace: a trust model", *2001 IEEE 35th International Carnahan Conference on Security Technology*, Oct 2001, p 235 –239

[291] Rodriguez, J., Martin, F., Noriega, Garcia, P., P., Sierra, C. 1998b. "Towards a test-bed for trading agents in electronic auction markets." *AI Communications,* 11(1):5-19, 1998.

[292] Rodriguez, J., Noriega, P., Sierra, C., Padget, J. 1998a. "Competitive Scenarios for Heterogeneous Trading Agents." *Proceedings of the Second International Conference on Autonomous Agents (Agents' 98).*

[293] Rosenschein, J. S. and Zlotkin, G. 1998. "Rules of Encounter. Designing Conventions for Automated Negotiation among Computers". MIT Press. ISBN 0262181592.

[294] Rosenschein, J., Sandholm, T., Sierra, C., Maes, P. and Guttmann, R.. 1998b. "Agentmediated electronic commerce: Issues, challenges and some viewpoints". *In Proceedings of the Second International Conference on Autonomous Agents*.

[295] Rothkopf, M., Pekec, A. and Harstad, R. 1998. "Computationally manageable combinatorial auctions". *Management Science*, 44:1131--1147, 1998.

[296] RuleML. http://www.dfki.de/ruleml

[297] Rushby J. 1994. "Critical System Properties: Survey and Taxonomy", *Reliability Engineering and System Safety, 43(2):* 189-219, 1994.

[298] Sadek, M. D. 1992. "A study in the logic of intention". *In Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA. pages 462--473.

[299] Samuelson L. 1998. "Evolutionary Games and Equilibrium Selection." *MIT Press ISBN: 0262692198*.

[300] Sandholm, T. 1993. "An implementation of the contractnet protocol based on marginal cost calculations." *Proc. Nat. Conf. On Artificial intelligence.* AAAI Press.

[301] Sandholm, T. 1996. "Limitations of the Vickrey Auction in Computational MultiAgent Systems". *In Proceedings of ICMAS-96*, pp. 299-306

[302] Sandholm, T. and Suri S. 2001. "Market Clearability". *In proceedings of International joint conference on Artificial Intelligence , IJCAI 2001*. Pages 1145 – 1152

[303] Sandholm, T., Sikka, S., and Norden, S. 1999b. "Algorithms for optimizing leveled commitment contracts". *In Proc. of IJCAI99*, pages 535-540.

[304] Sandholm, T.and Lesser, V. 1995. "Issues in automated negotiation and electronic commerce: Extending the contract net framework". *In 1st Int'l Conf. on Multiagent Systems*, pages 328--335, San Francisco.

[305] Sandholm, T.and Lesser, V. 1997. "Coalitions Among Rationally Bounded Agents". *Artificial Intelligence,* 94(1):99—137.

[306] Sandholm, T.W. 1999. "Distributed Rational Decision Making." *In Weiss, G. (ed.) 1999. "Mutiagent Systems A modern approach to distributed Artificial Intelligence".* MIT Press. ISBN 0262232030.

[307] Schild, K. 2000. "On the Relationship Between BDI Logics and Standard Logics of Concurrency2. *Autonomous Agents and Multi-Agent Systems* 3(3): 259-283

[308] Searle, J. R. 1969. "Speech acts: An essay in the philosophy of language". *Cambridge University Press.*

[309] Segev, A. and Beam, C. 1997. "Automated Negotiations: a Survey of the State of the Art". *In Wirtschaftsinformatik*, vol. 39, 1997, pp. 269-279

[310] Sen, S. 1996. "Adaptation, coevolution and learning in multiagent systems". *Papers from the 1996 AAAI Spring Symposium*. AAAI Press. AAAI Technical Report SS-96-01.

[311] Sen, S. and Weiss, G. 1999. "Learning in Multiagent Systems." *In Weiss, G. (ed.) 1999. "Mutiagent Systems A modern approach to distributed Artificial Intelligence".* MIT Press. ISBN 0262232030.

[312] Sen, S., Sekaran, M. and Hale, J. 1994. "Learning to Coordinate Without Sharing Information". *Proceedings of the National Conference on Artificial Intelligence*.

[313] Sergot M.J. 1999. "Normative Positions". *In Norms, Logics and Information Systems*, P. McNamara and H. Prakken (eds), IOS Press, Amsterdam, pp 289-308.

[314] Shanahan M.P. 1997. "Solving the Frame problem: a mathematical investigation of the Common sense law of Inertia". *MIT Press*, 1997

[315] Shanahan, M. 1997. "Event calculus planning revisited". *In Proc. of the European Conference on Planning (ECP),* volume 1348 of LNAI, pages 390-402. Springer, 1997.

[316] Shanahan, M. P. 1999. "The Event Calculus Explained". *In Artificial Intelligence Today*, eds. M. J. Wooldridge and M. Veloso, Springer-Verlag Lecture Notes in Artificial Intelligence no. 1600, Springer-Verlag, pages 409-430.

[317] Shannon, C.E. 1950. "Programming a Computer for Playing Chess". Philosophical Magazine, Vol. 41 (7th series), 256-275.

[318] Shehory O. and Kraus S. 1999. "Feasible formation of coalitions among autonomous agents in non-super-additive environments". *Computational Intelligence,* 15(3). 1999.

[319] Shoham, Y. 1991. "AGENT0: A simple agent language and its interpreter". *In Proceedings of the 9th National Conference on Artificial Intelligence*, Anaheim. 704-709.

[320] Shoham, Y. 1993. "Agent Oriented Programming", *Journal of AI , 60(1):51-92*

[321] Shoham, Y. and Tennenholtz, M. 1995. "On Social Laws for Artificial Agent Societies: Off-Line Design". *Artificial Intelligence*, 73, 231-252, 1995.

[322] Shostak R. E. 1984. "Deciding combinations of theories". *J. of the ACM*, 31(1):1-12.

[323] Sierra C., Faratin P. and N. R. Jennings. 1997. "A Service-Oriented Negotiation Model between Autonomous Agents". *Proc. 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, Ronneby, Sweden, 17-35, 1997.

[324] Sierra C., Godo, J., Lopez de Mantaras R., Manzano M. 1996. "Descriptive dynamic logic and its applications to reflective architectures." *Future Generation Computer Systems Journal, Elsevier*. 12:157-171.

[325] Sierra C., Jennings N.R., Noriega P., and Parsons S. 1998. " A framework for argumentation-based negotiation." *In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, Proc. ATAL-97*, pages 177-192. Springer-Verlag.

[326] Singh, M. P. 1998. "A Semantics for Speech Acts," in *Readings in Agents*. Edited by M. N. Huhns, and M. P. Singh. Morgan Kaufmann, San Francisco, 458-470.

[327] Singh, M. P., 1993. "A Semantics for Speech Acts". *Annals of Mathematics and Artificial Intelligence*.

[328] Singh, M. P., Rao, A. S. and Georgeff, M. P. 1999. "Formal methods in DAI: Logic-based representation and reasoning". *In G. Weiss, editor, Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence,* pages 331--376. The MIT Press.

[329] Sistla A.P. and Clarke E.M. 1985. "The complexity of propositional linear temporal logics". *Journal of the ACM*, 32(3):733--749, 1985.

[330] Sistla, A. P. 1994. "Safety, liveness and Fairness in Temporal Logic". *Formal Aspects in Computing,* 1994, vol. 6, pp 495-511.

[331] Smith, R. G. 1980. "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver". IEEE Trans. in Comp. 29 (12), 1104-1113.

[332] Spivey J. M. 1988. "The Z Reference Manual". *Prentice Hall International.*

[333] Stenning V. 1976. "A Data Transfer Protocol". *Computer Networks* 1.

[334] Stevens R S and Kaplan D J. 1992. "Determinacy of generalized schema". *IEEE Transactions on Computers* 41 (1992), 776-779.

[335] Sycara, K. 1990. "Persuasive argumentation in negotiation". *Theory and decision,* 28: 203-242.

[336] Tahara, Y., Ohsuga, A. and Honiden, S. 2001. "Mobile Agent Security with the IPEditor Development Tool and the Mobile UNITY Language". *Proceedings of 5th International Conference on Autonomous Agents 2001*. pp 656-662.

[337] Tambe, M. 1996. "Teamwork in real-world, dynamic environments". *In Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96),* Menlo Park, California, AAAI Press.

[338] Tambe, M. 1997. "Towards flexible teamwork". *Journal of AI Research*. 7:83-124.

[339] Tennenholtz, M. 2001. "Rational Competitive Analysis". *In International Joint Conference on Artificial Intelligence 2001*. pp 1067-1072.

[340] Tsang, E. 1993. " Foundations of Constraint Satisfaction". *Academic Press*, 1993

[341] Van der Hoek, W. and Wooldridge, M. 2002. "Model Checking Knowledge and Time". In *Proceedings of the 9th Int. SPIN Worshop. on Model Checking of Software*.

[342] Vidal, J. M. and Durfee, E. H. 1996. "The impact of nested agent models in an information economy". *In Proceedings of the Second International Conference on Multiagent Systems*. AAAI Press.

[343] Vinoski, S. 1997. " CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments". *IEEE Communications Magazine, Vol. 14, No. 2*.

[344] Wagner T,. Benyo, B., Lesser V., and Xuan P. 1999. "Investigating Interactions Between Agent Conversations and Agent Control Components". *In Agents 99 Workshop on Conversation Policies*.

[345] Wagner, D. and Schneier, B. 1996. "Analysis of the SSL 3.0 Protocol". *The Second USENIX Workshop on Electronic Commerce Proceedings*, USENIX Press, pp. 29-40.

[346] Waldinger, R. 1997. "Achieving several goals simultaneously". *Machine Intelligence 8*: *Machine Representations of Knowledge*, Chichester, England.

References

[347] Waldinger, R. J., Levitt, K. N. 1974. "Reasoning about programs". *Artificial Intelligence*, 5(3):235--316

[348] Walton, D. N. and Krabbe, E. C. W. 1995. "Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning ". *State Univ. New York P.*

[349] Watanabe, T., Mizobata, Y. and Onaga, K. 1989. "Time Complexity of Legal Firing Sequence and Related Problems of Petri Nets". Transactions of the Institute of Electronics, Information and Communication Engineers E, Vol. E72, No. 12, pages 1400-1409.

[350] Wang, X. F., Lam, K. Y. and Yi, X. 1998. "Secure Agent-mediated Mobile Payment*", In proceedings First Pacific Rim International Workshop on Multi-agents, Mutiagent Platforms.* Toru Ishida (ed.) Springer 3540659676.

[351] Warmer J. and Kleppe A. 1999. "OCL: The constraint language of the UML". *Journal of Object-Oriented Programming*, May 1999.

[352] Watkins, C. J. C. H. 1989. "Learning from delayed rewards." *PhD. Thesis, King's College*, Cambridge University.

[353] Webster, B F. 1995. "Pitfalls of Object-Oriented Development". *M&T Books*, New York, ISBN 1-55851-397-3.

[354] Weiss, G. (ed.) 1998. "Special Issue on Learning in Distributed AI Systems". *Journal of experimental and theoretical artificial intelligence*. Vol 10(3).

[355] Weiss, G. (ed.) 1999. "Mutiagent Systems A modern approach to distributed Artificial Intelligence". *MIT Press*. ISBN 0262232030.

[356] Weiss, G. 1993. "Learning to coordinate actions in multi-agent systems". *In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 311-316.

[357] Weiss, G., and Dillenbourg, P. 1999. "What is 'multi' in multiagent learning?" *In P. Dillenbourg (Ed.), Collaborative learning. Cognitive and computational approaches* (Chapter 4, pp. 64--80). Pergamon Press.

[358] Weld, D. S. 1994. "An introduction to least commitment planning". *AI Magazine* 15(4): 27-61.

[359] Wellman, M. P. 1995. "Market-oriented programming environment and its application to distributed multicommodity flow problems". *Journ. AI Res.* 1(1), 1-23.

[360] Wellman, M. P. 1998. "A Computational Market Model for Distributed Configuration Design". *Readings in Agents.* Michael N. Huhns and Munindar P. Singh (eds.) p371-379. ISBN 1-55860-495-2

[361] Wellman, M. P. and Wurman, P. R.1998. "Market-aware agents for a multi-agent world". *In Robotics and Autonomous Systems* 24:115-125, 1998

[362] White, J. E. 1994. "Telescript technology: The foundation for the electronic marketplace". *White paper, General Magic, Inc*., 2465 Mountain View, CA 94040.

[363] Winograd, T. and. Flores, F. 1986. "Understanding Computers and Cognition: A new foundation for design". *Addison Wesley*, Reading, MA 1986.

[364] Wooldridge, M. 1996. "Practical Reasoning with Procedural Knowledge: A Logic of BDI Agents with Know-How". *In D. M. Gabbay and H.-J. Ohlbach, editors,* International Conference on Formal and Applied Practical Reasoning. Springer-Verlag.

[365] Wooldridge, M. 1998. "Verifiable semantics for agent communication languages". *Third International Conference on Multi-Agent Systems*, Y. Demazeau (ed.), IEEE Press.

[366] Wooldridge, M. 2001. "An Introduction to MultiAgent Systems." *Wiley and Sons Ltd.* ISBN 0471-49691-X.

[367] Wooldridge, M. and Jennings, N. R. 1995a. "Intelligent agents: Theory and practice". *The Knowledge Engineering Review*, 10(2):115--152.

[368] Wooldridge, M. and Jennings, N. R. 1995b. "Agent Theories, Architectures, and Languages: A Survey." *In M. J. Wooldridge and N. R. Jennings, editors, Proc. ECAI-94 Workshop on Agent Theories, Architectures and Languages*. Springer-Verlag,

[369] Wurman, P. R., Wellman M. and Walsh W. 1998. "The Michigan Internet AuctionBot: A configurable auction server for human and software agents". *Second International Conference on Autonomous Agents*, May 1998.

[370] Yang, Q. 1997. "Intelligent Planning: A decomposition and abstraction-based approach" *Springer-Verlag, New-York.*

[371] Yang, Z., and Duddy, K. 1996. "CORBA: A Platform for Distributed Object Computing". *ACM Operating Systems Review*, Vol 30, No 2, PP 4-31, April 1996.

[372] Yi, X. and Okamoto, E. 1998. "A Secure Agent-based Payment System for Mobile Computing Environments", *IEICE Technical Reports*, Vol.98, No.84, 1998

[373] Yokoo, M. and. Ishida, T. 1999. "Search algorithms for agents". *In G. Weiss, editor, Multiagent Systems A Modern Approach to Distributed AI,* pages 165-199. MIT Press.

Acronyms

ABP	Alternating Bit Protocol
ACL	Agent Communication Language
AI	Artificial Intelligence
AMEC	Agent Mediated Electronic Commerce
ANML	Agent Negotiation Meta-Language
AOP	Agent Oriented Programming
ARCHON	ARchitecture for Cooperative Heterogeneous ON-line systems
AUML	Agent Unified Modeling Language
BDI	Belief, Desire, Intention
BOCA	Business Object Component Architecture
BOF	Business Object Framework
BSD	Business System Domain
CDL	Component Definition Language
CORBA	Common Object Request Broker Architecture
COS	Common Object Services
CPN	Coloured Petri Nets
CTL	Computational Tree Logic
CTL*	Computational Tree Logic*
EAUML	Extended Agent Unified Modeling Language
EC	Electronic Commerce
E-Commerce	Electronic Commerce

Acronyms

FIPA	The Foundation for Intelligent Physical Agents
FSM	Finite State Machines
IDL	Interface Definition Language
IP	Interaction Protocol
KB	Knowledge Base
KQML	Knowledge Query and Manipulation Language
MAS	Multi-Agent System
MASSIF	Mobile Agent Facility
OCL	Object Constraint Language
OMA	Object Management Architecture
OMG	Object Management Group
OOP	Object-Oriented Programming
ORB	Object Request Broker
OSM	Open Service Model
PDL	Propositional Dynamic Logic
SL	Semantic Language
SLA	Service Level Agreement
SSL	Secure Socket Layer
TCP/IP	Transport Control Protocol/ Internet Protocol
UML	Unified Modeling Language™
W3, WWW	World Wide Web
W3C	World Wide Web Consortium
XML	eXtended Mark-up Language