Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 134 (2005) 55–75
www.elsevier.com/locate/entcs

Propositional Statecharts for Agent Interaction Protocols

H. R. Dunn-Davies and R. J. Cunningham[1]

Department of Computing
Imperial College
London, UK
SW7 2BZ

S. Paurobally[2]

Department of Electronics and Computer Science
University of Southampton
Southampton, UK
SO17 1BJ

Abstract

Current representations of interaction protocols either concentrate on ease of understanding and neglect the need to specify protocols unambiguously (e.g. [2]), or concentrate on formal rigour and neglect usability issues (e.g. [21,10]). The on-going development of propositional statecharts has been motivated by the need to provide a notation which satisfies both of these requirements. Here we give a brief description of propositional statecharts, a restricted form of David Harel's statechart formalism [13] designed to represent agent interaction protocols. Statecharts are a popular extension of Euler Diagrams and are widely used as part of the UML framework. Propositional statecharts take advantage of the intuitive nature of the statechart formalism, and include labelling conventions to ensure that interaction protocols are defined unambiguously while remaining easy to understand.

Keywords: Agents, Agent Communication, Euler Diagrams, Higraphs, Propositional Statecharts, Statecharts, UML

[1] Email: hdd2@doc.ic.ac.uk, rjc@doc.ic.ac.uk
[2] Email: sp@ecs.soton.ac.uk

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2005.02.020

1 Introduction

At the turn of the millennium the field of software agent research had become one of the most active areas of R&D activity in computing [30]. Since then agents have been widely tipped as the logical next step in the evolution of computer science (e.g. [8]). Much of the intuitive appeal of the agent paradigm stems from the idea that agents could solve problems through cooperation and negotiation, as people do, therefore providing the potential for automating many of the tasks which currently still have to be performed by humans. The potential ability of agents to communicate with each other in a goal directed manner forms an integral part of this idea.

Researchers such as Greaves et al. [12] have identified three characteristics which must be shared by different agents to enable them to communicate effectively: a shared infrastructure for message passing, a common ontology, and a common agent communication language. These researchers have also noted that in all but the very simplest interactions it is necessary for each agent to have a common understanding of the set (or collection of sets) of rules underlying the interaction. A set of rules governing an interaction is commonly known as an Interaction Protocol.

The problems of ensuring that the agents involved in an interaction share a common infrastructure for message passing and a common agent communication language have been addressed by the design of general purpose frameworks for message passing (such as the ICM, described in [7]) and languages (such as KQML [9], or FIPA ACL, described at `http://www.fipa.org`[3]). However the design of a single generic protocol suitable for every interaction is less realistic. Agents may be expected to interact in a wide range of different circumstances, each of which may call for the use of different protocols. For example an agent may be required to communicate with a single agent, or with a group of agents at the same time, and could be required to participate in a tightly defined interaction such as an auction, which could take any one of a variety of forms. For this reason it would be extremely difficult to define a single protocol that would be suitable for every situation, making it necessary to find another way of ensuring that within any particular transaction, each agent is using exactly the same protocol.

Current agent interaction protocols are designed independently of agents that implement them. A library of such protocols can be found on FIPA's website (`http://www.fipa.org`). It is intended that agent designers implement these standard protocols when constructing software agents to enable

[3] FIPA is a non-profit organisation aimed at producing standards for the interoperation of heterogeneous software agents.

them to communicate with other agents effectively.

A key question for organisations such as FIPA that attempt to provide libraries of protocols for different agent designers to use is how to describe the protocols to the agent designers.

A number of notations have been proposed for this purpose, such as Petri nets [4] and modified finite state machines [10], but the most influential of these to date has been Agent UML [2]. The current notation used by FIPA for representing interaction protocols is the AUML sequence diagram taken from Agent UML.

UML [29] is a collection of diagrammatic notations that was introduced and standardised by Rational Software, Microsoft, and the Object Management Group during the 1990s and has become a standard method for designing object-oriented systems. Agent UML is an extension of UML which aims to apply the principles behind UML to the problem of agent oriented system design. There are a number of reasons why notations taken from Agent UML are a natural choice for the representation of interaction protocols. Their similarity to existing UML notations makes them easy to understand for agent designers, who would be familiar with UML notations already. This reduces the risk of agent designers choosing not to implement the standard protocols just because they have difficulty understanding the notation used to represent them. Another obvious advantage of UML based notations over other methodologies is the ability to benefit from the large amount of research which has already gone into the development of UML, and its proven track record as an effective means of designing complex systems.

Despite the obvious benefits of using the Agent UML suite for representing agent interaction protocols, there is an important problem with them that has yet to be resolved. This problem stems from a key difference between conventional software engineering and protocol design and implementation.

In conventional Software Engineering, UML has been used to provide a relatively abstract description of a system giving an overview of how it would work. In this way the UML diagram has acted as a useful reference for the programmer. The object constraint language (OCL) included in the UML framework provided a means of introducing formal constraints, however it was designed for object modelling and addressed only preconditions and post-conditions [28]. The exact details of how a system would be implemented were intentionally left out for two reasons. Firstly, the inclusion of every detail could compromise the clarity and simplicity of the representation, making it less useful for the programmer rather than more useful. Secondly, when initially designing a system it is not the intention to give a complete and unambiguous specification of it, as it would be unreasonable to expect a designer

to be able to do this for all but the simplest of systems. In practice UML models have given a useful informal description of a system rather than a precise specification and the programmer has been expected to fill in the local detail as he/she goes along [4].

Agent Interaction Protocols are often implemented separately in different agents by different programmers. Despite this each different agent's interpretation of a given protocol must be essentially the same in order to eliminate misunderstandings between agents. For this reason any adequate notation used to specify common Agent Interaction Protocols must provide an unambiguous representation of the protocol being described. It is no longer adequate to provide an informal description. Note that we include within the loose term unambiguous the property of logical completeness described by Paurobally and Cunningham [25].

Paurobally and Cunningham [24] have shown that, despite being extended for the purpose of Agent Oriented software engineering, Agent UML notations lack the precise action semantics required to provide an unambiguous representation of interaction protocols, and are better suited for informal descriptions of Interaction Protocols. A more rigorous description, which would force the protocol designer to specify the protocol exactly, is required. Paurobally [21] has developed one such notation, ANML, which is essentially an extended form of dynamic logic [15], but this notation is heavily logic based, and would therefore be of limited explanatory appeal to agent designers. Another notation, based on finite state machines, has been proposed by Fornara and Colombetti in [10], but this notation also lacks the simplicity and clarity of less rigorous forms of representation such as Agent UML.

The development of propositional statecharts has been motivated by the need to provide a notation which is sufficiently intuitive and easy to understand to encourage agent designers to adopt standard protocols while preserving the formal rigour required to represent the protocols unambiguously. It is also intended to form the basis of further work on the automated conversion of protocols described using a propositional statecharts into dynamic logic, which may be verified, and also into executable program threads. In this paper we will describe the propositional statechart formalism and demonstrate its ability to provide simple yet formally rigorous descriptions of the English Auction Protocol. The English Auction Protocol is a natural choice of interaction protocol to use for this purpose because it is well known and widely used

[4] The latest version of UML is designed to enable more formal descriptions, and a number of tools have been designed which can generate code from specialised UML diagrams in a number of application domains [18]. The current work is intended to facilitate the design of similar tools for the automated generation of executable interaction protocols described using propositional statecharts.

in e-commerce. For example the auction site eBay uses this protocol [6]. It is also surprisingly complex, and forms an interesting example of an iterative Interaction Protocol [10]. We will conclude by discussing opportunities for further work.

2 From Euler Diagrams to Statecharts

The propositional statechart formalism presented here is based on David Harels statecharts [13]. These are based on higraphs [14], which are a modified form of Euler diagrams described in [20]. An Euler diagram is a collection of closed curves known as contours arranged on a 2D plane, which is divided into separate regions corresponding to the areas of the plane inside different collections of contours. A similar principle underlies Venn diagrams (described in [17]).

We can recall here that a particular quality of Euler diagrams which makes them an ideal foundation for the statechart formalism is their ability to represent hierarchies. This gives rise to a more economical way of describing transitions to or from groups of states than is possible using conventional state transition diagrams, making the statechart formalism potentially more expressive [5].

The incorporation of Euler diagrams into higraphs involved two main modifications to their definition [14]. Firstly, higraphs required unique contours to be added to represent all identifiable sets. So while a conventional Euler diagram might use two overlapping contours A and B enclosed in a single contour U to represent four distinct areas ($A \cap B, A - (A \cap B), B - (A \cap B)$, and $U - (A \cup B)$), the modified higraph definition would require the inclusion of four further atomic contours (which do not contain any others), one in each of the areas specified. So in figure 1 a diagram consisting only of the contours labelled U, A and B would be a valid diagram, but the addition of 4 further contours such as those labelled C, D, E, and F in the diagram is necessary for the diagram to be a valid higraph. This modification allows the different areas to be labelled more clearly, as they each have their own contour dedicated to them [14].

Secondly, higraphs modify Euler diagrams to allow the representation of Cartesian products. A Cartesian product is represented by dividing a region enclosed by an individual contour into orthogonal parts using a dashed line.

[5] We say *potentially*, because while a hierarchy of states with predetermined finite depth can be replaced by a flat finite state diagram, this is no longer possible with a recursive hierarchy of unbounded depth. Such recursive hierarchies must be allowed if a visual formalism is not to be restricted in the computational processes it can represent.

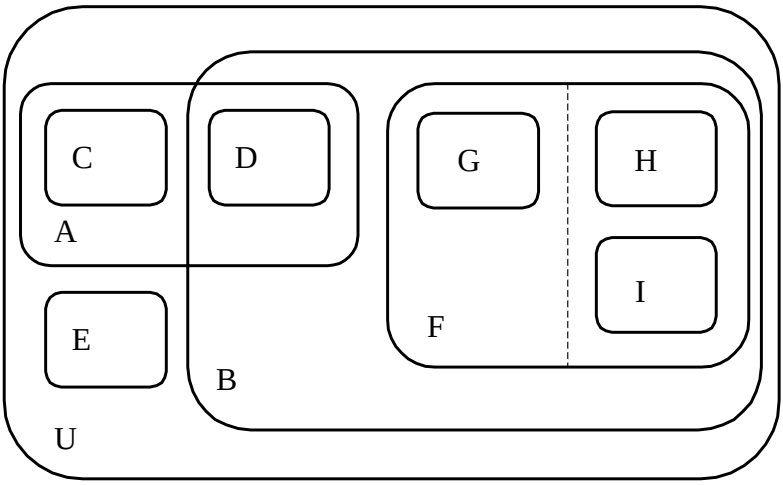

Fig. 1. An Euler diagram containing unique contours for all non-empty sets and displaying both overlapping contours and orthogonality

The region labelled F in figure 1 is an example. The set represented by an orthogonally divided region is the Cartesian product of the sets on either side of the dashed line. This property enables statecharts in the form of higraphs to represent concurrent systems without having to represent different combinations of states individually.

The diagram in figure 1 could be used to provide an economical description of the states of a system, with each contour representing a possibly hierarchical state, and topological enclosure determining the hierarchy. However the modified form of Euler diagrams described above is not sufficiently expressive to describe state transitions. In order to do this higraphs also incorporate directed hypergraphs. Hypergraphs [3] are an extension of discrete graphs whereby edges in the graph are not restricted to representing binary relations, and as such can connect sets of nodes as well as single nodes, as shown in figure 2. In a directed graph edges signify one way relations, and are drawn as arrows to signify their direction. Figure 2 shows a simple directed graph, in which one node is connected to a second by means of a single edge, alongside a directed hypergraph, in which two nodes are connected to a third by means of a single complex edge.

Directed hypergraphs are incorporated into higraphs through the inclusion of directed edges between contours. Figure 3 shows a higraph comprising the euler diagram in figure 1 with some edges added. The edges allow us

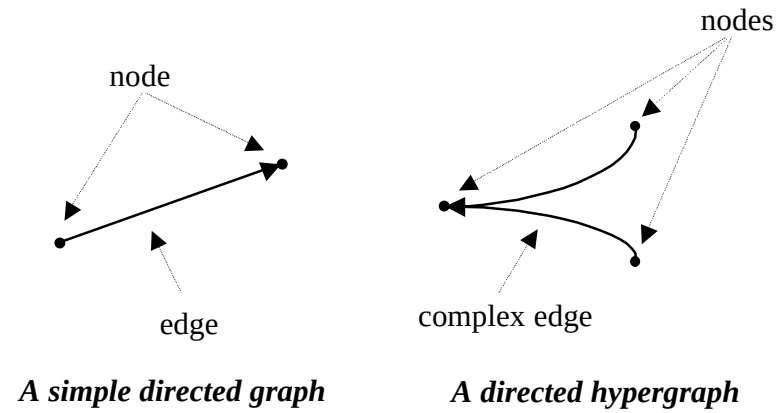

Fig. 2. A Directed Graph and a Directed Hypergraph

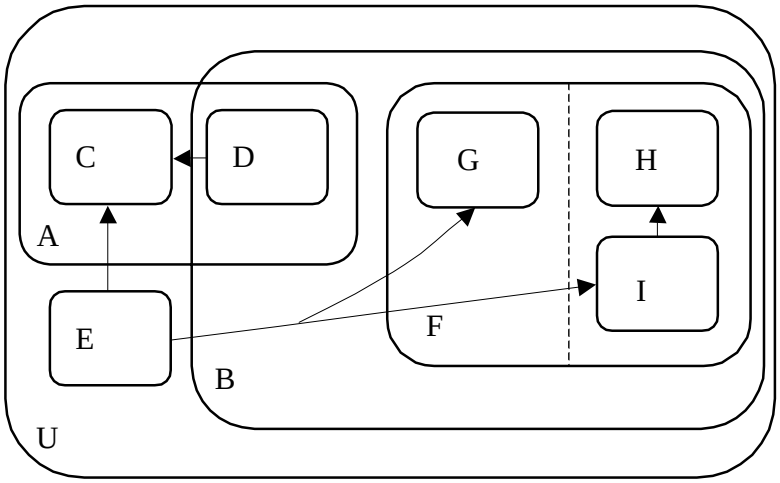

Fig. 3. The higraph formed by adding edges to the Euler diagram in figure 1

to describe one way relations between contours or groups of contours. This capacity makes them well suited to the representation of state transitions, which are referred to in this paper as events.

While UML statecharts are based on higraphs they do contain a number of features which are not included in basic higraphs. These are described by Harel and Politi [16]. Perhaps the most important addition is that of edge

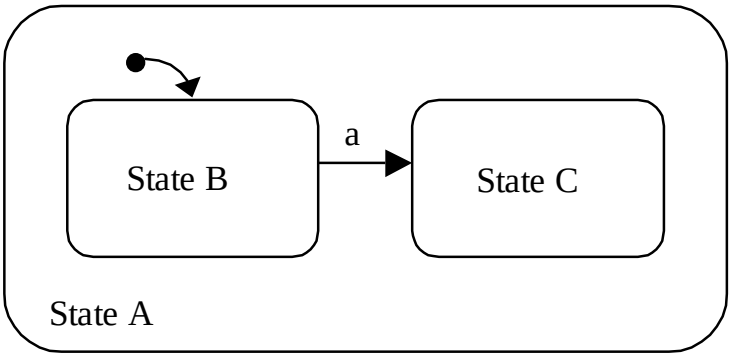

Fig. 4. A simple statechart with a default start state

labelling. In a statechart each edge has a label associated with it. This label defines the event that triggers the transition represented by the edge.

Another significant addition is the default start state, which allows the designer to specify the initial state of the system and its subsystems. Figure 4 gives an example of a default start state. The black dot with an arrow pointing to state B specifies state B as the default start state when state A is entered. Without this feature the statechart designer would not be able to explicitly specify the start state of a system or subsystem.

A third key feature of statecharts is the implicit assumption that atomic states whose contours are not directly contained in an orthogonal region are mutually exclusive (here we use the term atomic state to refer to any state with no substates). This assumption, combined with the requirement that higraphs include unique contours for all identifiable sets, means that statecharts provide unambiguous representations of the states of a system. The syntax and semantics of statecharts are discussed in depth in [16].

There are a number of reasons why we have chosen to base our graphical methodology on statecharts. Firstly, since they are part of the UML suite, they share the benefits of other methodologies derived from UML, which are described in section 1. Secondly, the inclusion of state hierarchy enables them to provide simpler, more economical representations of protocols than other graphical methodologies such as finite state machines [16]. Finally, a modified form of statecharts could provide an unambiguous representation of interaction protocols. This would facilitate consistent implementations of the protocol in

different agents.

3 Propositional Statecharts

The propositional statechart formalism introduced here aims to preserve the benefits of using statecharts to represent interaction protocols while strengthening their ability to do so unambiguously. In order to achieve this we have made a number of modifications to the statechart formalism. Firstly we have simplified the definition of statecharts by removing orthogonality. Secondly, we have introduced labelling conventions that are intended to be sufficiently intuitive to render the propositional statecharts easy to understand, yet restrictive enough to help designers provide unambiguous representations of protocols. Both of these modifications are described here in detail.

3.1 The Absence of Orthogonality

While many of the elegant features of statecharts, such as their ability to efficiently represent hierarchies, have been preserved, one important feature - the ability to represent orthogonality - has been removed. Orthogonality was included in UML statecharts in order to enable them to provide an economical description of complex reactive systems made up of relatively independent subsystems. In such a system the number of states the system, as a whole, can be in is approximately the product of the number of states each subsystem can be in. Without orthogonality each of these states would have to be drawn as a separate contour on the statechart. In contrast, using orthogonality only the states of each subsystem need be represented as contours. For a system which is made up of a large number of subsystems which could each be in a number of states the use of orthogonality leads to a far more economical description than would otherwise be possible. This is illustrated in figures 5 and 6, which show the same system represented by two statecharts, the first displaying orthogonality and the second none.

When a statechart is used to represent an interaction protocol it is possible to consider each state of the interaction as a combination of the states of each agent involved in the interaction, and so decompose an interaction protocol into a series of orthogonal sub-protocols, each representing the possible states of a particular agent. There are two reasons for not approaching this task using the orthogonality mechanism of standard statecharts. Firstly the agents themselves are not subsystems, it is only their sub-protocols which must combine, so we have the conceptually tricky problem of combining parts of interactions between temporally autonomous agents into a pseudo whole. Secondly in a typical interaction protocol the agent states are far from inde-

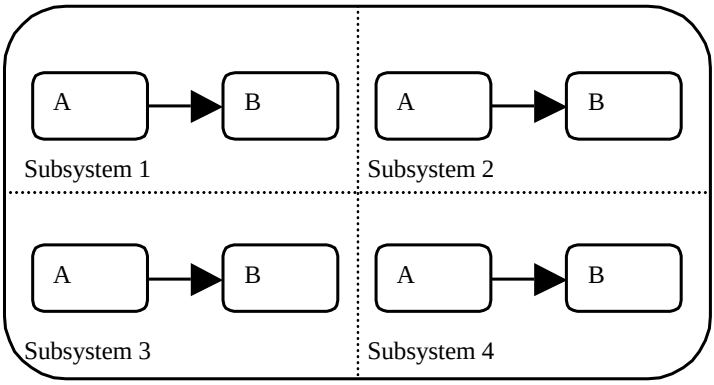

Fig. 5. A Simple Statechart with orthogonality

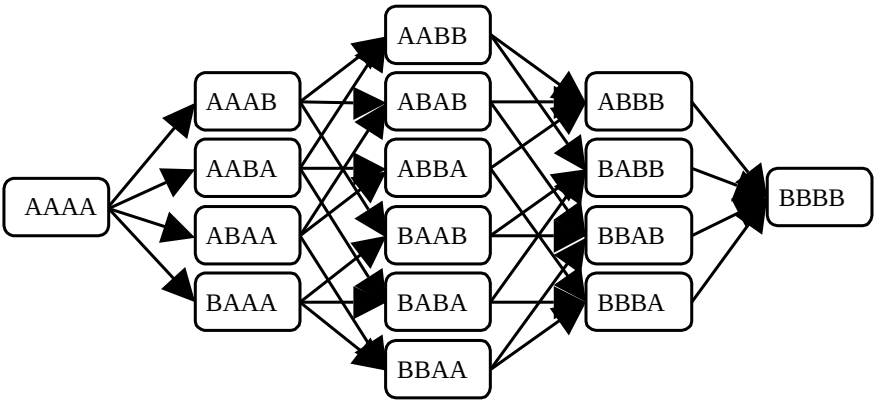

Fig. 6. Figure 5 without orthogonality

pendent, as many or all of the agents may be in the same protocol state at any particular time, or may be following a similar sub-protocol. In these cases the use of orthogonality would provide little benefit for the representation of states. Instead, one can represent each sub-protocol using a separate, and often similar, or generic, statechart. The absence of orthogonality simplifies the propositional statechart formalism. We then seek to clarify the verification and implementation of protocols by considering the properties of these simpler charts, and how they interact.

3.2 Labelling Conventions

As well as simplifying Harel's statecharts, we have added new features to strengthen the statechart language as a means of representing interaction protocols. These take the form of rules which govern the labelling of states and events. These labelling conventions motivated the decision to name our modified statecharts propositional statecharts.

3.2.1 State Labels

A state label is an identifier for a state. An effective state labelling system should enable the user of the statechart to refer to each state unambiguously. To achieve this goal state labels should be assigned so that if the statechart were flattened out, and the names of the superstates were appended to those of the substates, no two states would have the same label.

Aside from this basic requirement, any labelling conventions for the states themselves are purely for the benefit of anybody wishing to inspect the protocol by looking at the statechart. For this purpose it is useful to give an accurate if incomplete description of the current state of the negotiation. The convention we use to name the states is to describe them using predicates, but expressed in terms of the action or actions either directly preceding or directly following them within the same superstate. In this way the state labels give a rough indication of which actions have occurred, enabling anyone looking at the propositional statechart to understand the protocol being represented more easily. The state labels in figures 7, 8, and 9 have been created according to this convention.

One possible benefit of a convention such as this one, which standardises the labelling of states, is that clear and consistent state labels could to a large extent be created automatically using a graphical tool designed to draw propositional statecharts for interaction protocols.

In order to provide a complete description of Interaction Protocols such as the variants of the English Auction Protocol described in this paper it is

necessary to include variables to represent, for example, the highest bidder, whose value can change throughout the interaction. Furthermore, in order to provide an unambiguous representation of the protocol it is necessary to specify the scope of these variables. This can be done by including a single hierarchical state which encloses all of the states and actions in which the variables are required, and by specifying the variable names as part of that state label. An example of this is shown in figure 8, where the largest state is the superstate of all of the other states and its state label includes the names of the three variables associated with the interaction.

3.2.2 Event Labels

An event label describes an event; a transition from one state to another, denoted by a directed, hypergraph-edge between the states. The inclusion of hypergraphs, coupled with the ability to represent hierarchies, means that the statechart designer can choose whether to depict each transition between concrete states using a separate edge, or to depict transitions common to a number of concrete states by means of a single complex edge at a higher level in the state hierarchy. For example, in figure 8, the edge labelled *auctioneer gives up* represents a number of possible transitions, one for each of the concrete states included in its initial state (labelled *more than one bidder accepted*). Besides providing an identifier for such a transition, a statechart event label also needs to describe the conditions under which a transition occurs. For an agent interaction protocol each event may describe any agent action which can trigger the relevant transition, and may include auxiliary pre-conditions and post-conditions. An interaction event may involve many possible agents, which are categorized by their role. The event labelled *more than one bidder accepts* in figure 8 is an example of this. The number of participants a protocol can handle in each role can be inferred from the event labels. For example, in figure 9 we can infer that there is only one auctioneer from the fact that the auctioneer is referred to directly without the use of a quantifier. In the case of an auction the user of the protocol may wish to place further constraints on the number of participants (for example by limiting the number of bidders allowed). This could be achieved by requiring that each agent register to take part in the interaction, and limiting the number of agents that could register for any particular role. This strategy is often employed in e-commerce [22].

While an event can be simply depicted as a transition between abstract protocol states, its definition, in terms of atomic conditions and actions, can be complex. To include complex event labels to describe these transitions would be contrary to the purpose of a diagrammatic notation which aims to be clear and easy to understand. However, the complexity of event expressions can be

directly alleviated by choosing a simple identifier to label the transition, with an accompanying definition. This method may be considered desirable for the complex dynamic logic path labels in figure 9, although we have not used it here.

In their book Modeling reactive systems with statecharts [16], Harel and Politi propose the introduction of activity diagrams to give a dual hierarchy for information flow through processing activities. This perspective has not persisted because an information processing activity can also be regarded as a discrete, although possibly compound processing state, and a process as a path between such processing states in program execution. While a primitive interaction event may be the action of an agent in sending a message, a more complex interaction can be expressed as the composition of primitive actions in the way primitive programs are composed in a sufficiently rich dynamic logic. Thus we treat an event in a propositional statechart as expressed by the composition of actions by agents in certain roles, including test actions which capture and generalize the auxiliary pre- and post-conditions of statecharts. The language we have used elsewhere for expressing the events in protocols is derived from ANML [21], and allows us to extract from the interaction statechart a formal theory in the dynamic logic [15].

For the purpose of this exposition we take an eclectic view by illustrating both informal, and more formal variants of event labels. In each case the agent is specified using quantification with a role restrictor, as in *some bidders bid* (figure 7), and *some bidder b accepts* (figure 9). Note that figures 7 and 8 do not represent the same protocol.

In figure 9, the event label syntax is program-like, and adapted from dynamic logic. It includes operators for sequential composition (;) and conditional action (?), as well as Boolean connectives ($\vee$, $\wedge$, $\neg$), and informal quantifiers.

3.3 Definitional Matters

We do not include a formal definition of either syntax or semantics, because there are unresolved issues, most obviously regarding the acceptable forms of logical quantification in path labels. It is clear from the foregoing discussion that the core syntax of a propositional statechart is intended to be a restriction of conventional statecharts, so that the abstract syntax of contours as nodes in a hierarchy, and of paths as labelled links between nodes should be similar. Nor is there evident advantage in not borrowing tested visual representations, as concrete syntax for the core abstract syntax.

The intended semantics of a propositional statechart its re-expression as a theory in a definitional syntax of first order dynamic logic. There are three

key elements to this interpretation:

(i) Each state label is interpreted as a proposition, a propositional variable in the case of an atomic state, as a predicate with parameters (attribute variables) in the case of a contour with substates.

(ii) In the case of states at the same hierarchical level, each can be related to others through the action modalities of the dynamic logic in much the same way that graph algebras can be used to represent finite state machines.

(iii) In the case of a state hierarchy, the proposition expressed by a contour state is defined (with appropriate quantification) as an exclusive disjunct of its immediately contained state propositions. (This re-interprets a key insight of Harel's statechart formalism, that distinct non-orthogonal states are mutually exclusive).

This interpretation appears to provide key elements a logical theory that is amenable to the methods of computational logic, and indeed it arises from our previous attempts to clarify imperfect statechart-like descriptions of interaction protocols which have arisen in the work of industrial associations such as FIPA. Of course, such a theory needs an operational semantics as well. But dynamic logic, although intimidating to those unfamiliar with modal logics, is a more expressive generalisation of so-called "program logics", with denotational (Kripke) semantics, and computational interpretations.

An important motivation for our approach is the need to relate the mutual beliefs of agents when each takes a participating role in a protocol. This problem is illustrated by the decomposition of figure 9, as a common protocol for an English auction, into a separate thread for the auctioneer (figure 10), and for each bidder (figure 11). A key step that we have taken is to embody the communicative acts *bid* and *announce* within local message actions SEND and RECEIVE. But here too we have illustrated our intent, not the rules which may govern such decomposition.

4 Representing the English Auction

Paul Milgrom [19] defines the English auction in the following way. "Here the auctioneer begins with the lowest acceptable price [perhaps] the reserve price and proceeds to solicit successively higher bids from the customers until no one will increase the bid[6]. The item is 'knocked down' (sold) to the highest bidder."

[6] We have inserted [perhaps], because it is manifestly the case that the auctioneer in an English auction does not necessarily start with the reserve price, even when there is one.

This is a loose definition, and in practice there are many variations in everyday use. Many, if not most, do not begin with a reserve price. Figure 7 is a naive and incomplete depiction of an English auction, but one that many will recognize. It does not have a reserve price, and the challenge of making it into a specification which can be executable by a machine leads to contentious alternatives, partly because most programmers are used to thinking in terms of discrete time steps, in which many bids may occur, so that there is a problem in managing selection. The depiction of the protocol in figure 7 leaves open the question of how the auctioneer deals with a situation where there is more than one highest bidder. A more detailed version of the protocol, shown in figure 9, resolves this by specifying that the auctioneer accepts the bid made by just one of the highest bidders, possibly arbitrarily, and it is only through the process of one bid being accepted by the auctioneer that a partial action path to a new bid state is composed. This can really only be represented using a path label. In figure 9 the auctioneer also announces to all bidders the price that has been bid and the identity of the highest bidder. This announcement is critical if real bids are to be allowed. Thus figure 9 is closer to real life, with independent agents, than figure 7, but its events are more sophisticated.

Electronic variants which occur on well known auction sites tend to start with a reserve price (the logic is slightly simpler with a reserve price) and then to receive bids. This leads to a three way choice of action for the auctioneer, depending on whether there are no bids, just one bid, or more than one bid. The variant of the English Auction Protocol in figure 8 illustrates this sort of refinement. Furthermore, it is now natural for the auctioneer to be responsible for suggesting a new price each time a bid is made. The bidders merely have to specify that they are willing to pay the latest price suggested by the auctioneer. Secondly, the auctioneer has the option to give up and declare the item unsold at any stage in the auction once it has started. This ensures that the auction can always end even if there is no single highest bidder. However, a flaw arises when more than one bidder agrees to pay a given price, but no bidders agree to pay the next price proposed by the auctioneer. In the auction as it is specified here the auctioneer has no option but to give up, however in real life the auctioneer would be unlikely to declare the item unsold, and may decrease the price slightly, or choose a bidder at random, in an attempt to sell it.

There are two state variables associated with the versions of the English Auction Protocol shown in both figure 9, these are denoted *cp*, and *hb*, and stand for ‘current price‘ and ‘highest bidder‘ respectively. Values are assigned to them through explicit assignment as part of the action path events.

The English Auction described in figure 9 does not require any messages to be received by some agents involved in the interaction but not others. How-

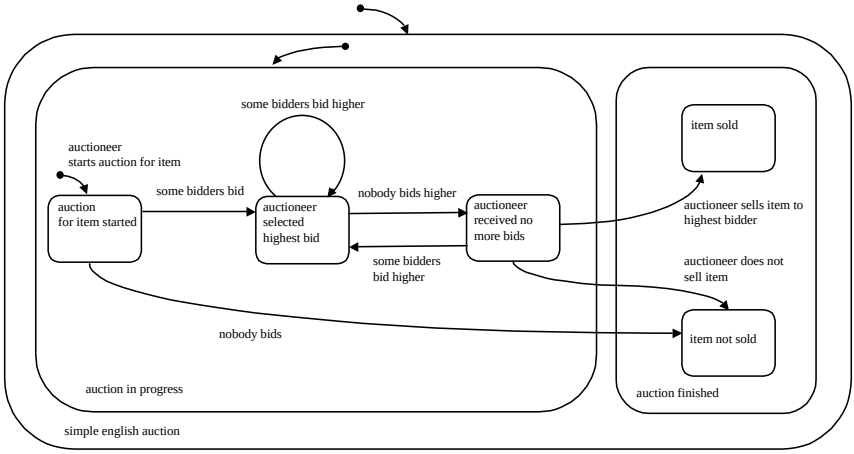

Fig. 7. A simple statechart for an incompletely specified English Auction Protocol

ever, one can imagine a variation on the English Auction which involves a private reserve price, which must be declared to a third party by the auctioneer but cannot be revealed to the bidders. In such a case it would be necessary to conceptually divide the auction into two separate interactions, a main interaction representing the part of the auction which involves all of the agents, and a sub-interaction representing the private sequence of messages passed between the auctioneer and the third party.

5 Conclusions and Future Directions

Representations of Interaction Protocols for Agent Interactions have traditionally either concentrated on ease of understanding and neglected the need for formal rigour and unambiguous protocol specification (e.g. [2]), or have concentrated on formal rigour and neglected usability issues (e.g. [21,10]). In this paper we have described the Propositional Statechart formalism, a modified form of David Harel's highly successful statechart formalism [13]. By basing our representation on Harel's statechart formalism we have ensured that it is easy to use and understand, and provides an intuitive form of representation for agent designers, many of whom are already familiar with statecharts in the

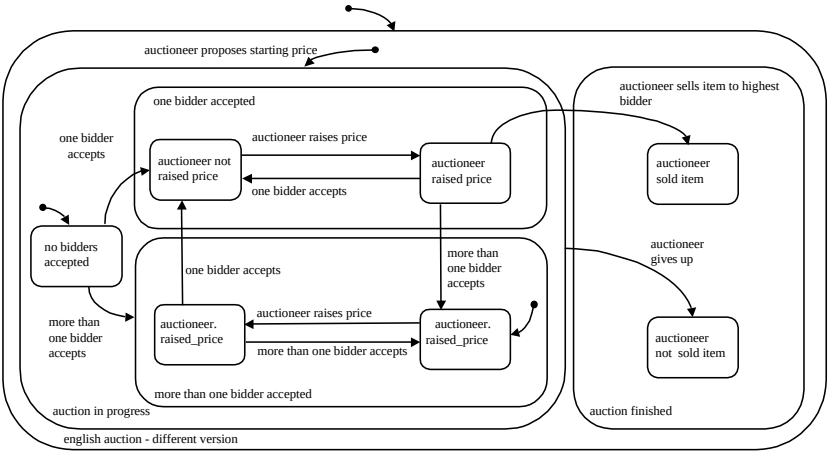

Fig. 8. A simple statechart for a different version of the English Auction Protocol where the auctioneer proposes prices and multiple acceptance leads to different states

context of UML.

We have suggested some modifications to statecharts for use with interaction protocols, aiming to preserve the intuitive nature of statecharts while ensuring that interaction protocols specified using the modified form of statecharts can be specified clearly and unambiguously. Propositional statecharts include labelling conventions for events which specify the conditions under which the events can occur. In general this seems to require path labels in the style of dynamic logic, because intermediate states are not always reified. However it also appears that the logic of more conventional statecharts can be enriched by associating program variables with states and using preconditions and postconditions for implicit consulting and updating. Orthogonality is not an efficient way to represent interaction protocols, so the propositional statecharts we use are simpler than the standard variants.

The work presented here is also a foundation for two other technical developments. The first is work to support verification of interaction protocol properties, such as soundness and completeness [23]. The choice of labelling conventions proposed here has been motivated partly by the desire to enable a direct translation between propositional statecharts and an extended form of dynamic logic in which such logical properties can be analysed. The sec-

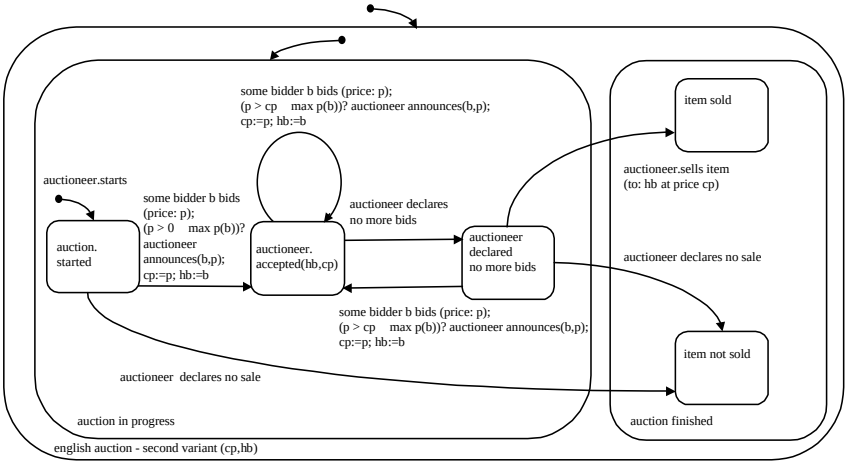

Fig. 9. A more detailed version of the English Auction Protocol shown in figure 7, with agent/action path event labels

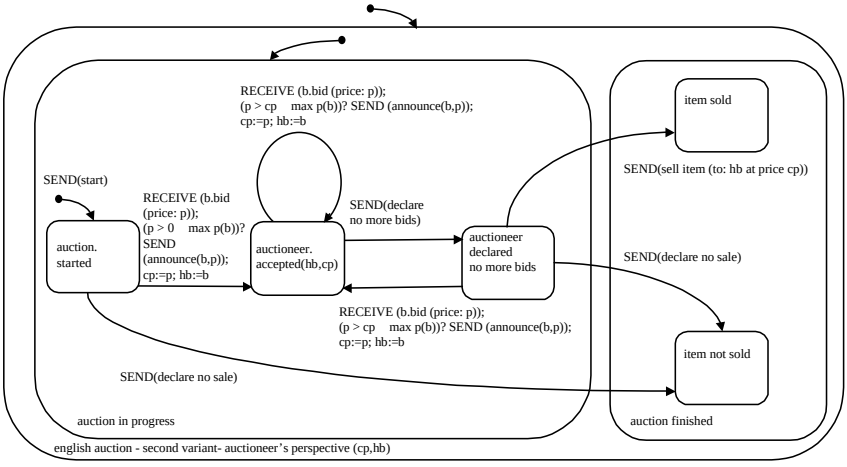

Fig. 10. The protocol shown in figure 9 from the point of view of the auctioneer

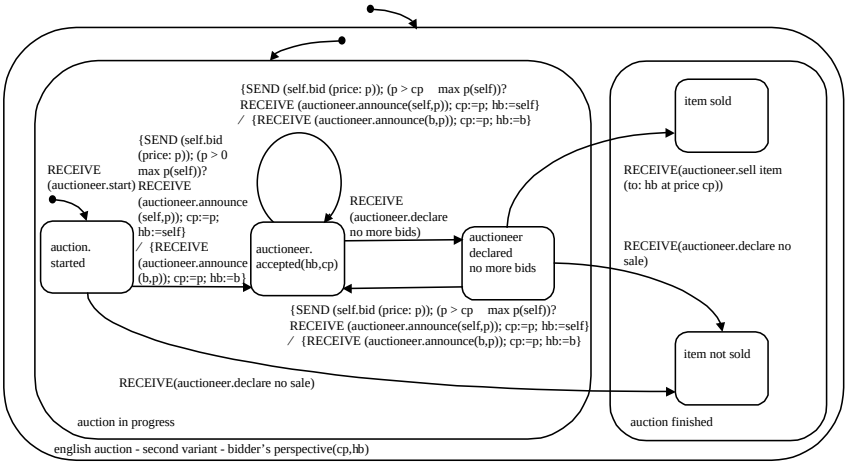

Fig. 11. The protocol shown in figure 9 from the point of view of a bidder

ond development (illustrated by figures 9-11) is that propositional statecharts, particularly with dynamic logic path labels, also seem to provide a foundation for the conversion of Interaction Protocols into a set of executable program threads. The provision of a library of executable threads for a given interaction protocol, which could be distributed and used by different agents would be a significant step towards the goal of eliminating misunderstandings between agents.

For these goals to be met there is more work to be done in a number of areas: tool support for reasoning about a sufficiently rich class of protocols for credible applicability, usability testing to ensure that a propositional statechart methodology can be accepted by agent developers, and practical solutions for imperfect situations like communication breakdown. In real world applications such as those involving mobile agents, or wireless communication, the ideal of perfect communication cannot be attained [27]. The consequences of relaxing assumptions which constitute an ideal could present a serious hurdle in moving from theory to practice.

References

[1] Austin, J.L., "How to Do Things With Words," Cambridge : Harvard University Press, 1975.

[2] Bauer, B., J.P. Muller, and J.Odell, *Agent UML: A Formalism for Specifying Multiagent Software Systems*, AOSE (2000), 91-104.

[3] Berge, C., "Graphs and hypergraphs," North-Holland Mathematical Library 6, 1973.

[4] Cost, R., Chen, Y., Finin, T., Labrou, Y., and Peng, Y, *Modeling agent conversations with colored petri nets*, Workshop on Specifying and Implementing Conversation Policies (1999), 59-66.

[5] Covington, M.A., "Natural Language Processing for Prolog Programmers," Prentice-Hall, 1994.

[6] David, E., Azoulay-Schwartz R. and Kraus, S., *An English Auction Protocol for Multi-Attribute Items*, Revised Papers from the Workshop on Agent Mediated Electronic Commerce IV, Designing Mechanisms and Systems (2002).

[7] Dale, J., "ICM Pocket Reference, " Fujitsu Laboratories of America Technical Memorandum FLA-NARTM99-02, Fujitsu Laboratories of America, 1999.

[8] Faltings, B., *Intelligent Agents: Software Technology for the New Millennium*, Informatik, 2000.

[9] Finin, T., Labrou, Y. and Mayfield, J., *KQML as an Agent Communication Language*, Software Agents, Jeff Bradshaw (Ed) (1995), 267-273.

[10] Fornara, N. and Colombetti, M., *Defining Interaction Protocols using a Commitment Based Agent Communication Language*, Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (2003).

[11] Foundation for Intelligent Physical Agents," FIPA Agent Communication Language Specification", URL: http://www.fipa.org.

[12] Greaves M., Holmback, H. and Bradshaw J., *What is a conversation policy?*, Issues in Agent Communication (2000).

[13] Harel, D., *Statecharts: A visual formalism for complex systems*, Sci. Comput. Programming **8** (1987), 231-274.

[14] Harel, D., *On visual formalisms*, Comm. Assoc. Comput. Mach. **31(5)** (1988), 514 -530.

[15] Harel, D., Kozen D. and Tiuryn, J., "Dynamic Logic," MIT Press, 2000.

[16] Harel, D. and Politi, M., "Modeling reactive systems with statecharts," McGraw-Hill, 1998.

[17] Howse, J., Molina, F., Taylor, J., and Shin, S., *Type syntax and token syntax in diagrammatic systems*, Proceedings of the international conference on Formal Ontology in Information Systems (2001), 174 185.

[18] Mellor, S., and Balcer, M., "Executable UML: A Foundation for Model Driven Architecture," Addison Wesley, 2004.

[19] Milgrom, P., *Auctions and bidding: A primer*, Journal of Economic Perspectives **3** (1989), 322.

[20] Mutton, P., Rodgers, P., and Flower J., *Drawing graphs in euler diagrams*, Diagrams 2004, LNAI **2980** (2004), 66-81.

[21] Paurobally, S., "Rational Agents and the Processes and States of Negotiation, " Ph.D. thesis, Imperial College, London, 2002.

[22] Paurobally, S. and Cunningham, R.J., *Specifying the processes and states of negotiation*, Agent Mediated Electronic Commerce. The European AgentLink Perspective, Springer-Verlag (2000), 61-77.

[23] Paurobally, S. and Cunningham, R.J., *Safety and liveness of negotiation protocols*, AISB2002 Intelligent Agents in virtual market track (2002).

[24] Paurobally, S. and Cunningham, R.J., *Verification of protocols for negotiation between agents*, ECAI **15** (2002), 43-48.

[25] Paurobally, S. and Cunningham, R.J., *Achieving common interaction protocols in open agent environments*, Proc. AAMAS03 Agentcities Workshop: Challenges in open agent environments (2003).

[26] Paurobally, S., Cunningham R.J. and Jennings, N.R., *Developing agent interaction protocols using graphical and logical methodologies*, Work. on Programming MAS, AAMAS (2003).

[27] Paurobally, S., Cunningham R.J. and Jennings, N.R., *Ensuring consistency in joint beliefs of interacting agents*, 2nd Int. Joint Conf. on Autonomous Agents and Multi Agent Systems (2003).

[28] Richters M. and Gogolla M., *On formalizing the UML object constraint language OCL*, Conceptual Modeling, LNCS 1507, Springer, (1998), 449-464.

[29] Rumbaugh, J., Jacobson, I., and Booch, G., "The Unified Modeling Language Reference Manual," Addison Wesley Professional, 1999.

[30] Woolridge, M. and Ciancarini P., *Agent Oriented Software Engineering: The State of the Art*, Agent-Oriented Software Engineering, First International Workshop, AOSE 2000, Revised Papers, LNAI **1957** (2000).