An Executable Specification of an Argumentation Protocol

Alexander Artikis[1], Marek Sergot[2] and Jeremy Pitt[1]
[1]Imperial College London, Electrical and Electronic Engineering Department, London SW7 2BT, UK
[2]Imperial College London, Department of Computing, London SW7 2BZ, UK
a.artikis@ic.ac.uk, mjs@doc.ic.ac.uk, j.pitt@ic.ac.uk

ABSTRACT

Open multi-agent computational systems are composed of heterogeneous and possibly antagonistic software entities. Characteristic features are limited trust and unpredictable behaviour. Members of such systems may fail to, or even choose not to, conform to the norms governing their interactions. It has been argued that systems of this type should have a formal, declarative, verifiable, and meaningful semantics. We present a theoretical and computational framework being developed for the executable specification of such systems. We adopt an external perspective and view open computational systems as instances of normative systems. In this paper we demonstrate how the framework can be applied to specifying and executing an argumentation protocol based on Brewka's reconstruction of Rescher's theory of formal disputation. The specification is formalised in the action language $C+$ and executed using the 'Causal Calculator' (CCALC) implementation.

1. INTRODUCTION

A particular kind of Multi-Agent System (MAS) is one where members are developed by different vendors and do not necessarily have a notion of global utility—indeed they may well be in direct competition with one another. MAS of this type are often classified as 'open' [9, 5, 18]. Examples include electronic marketplaces [1], automated negotiation and dispute resolution services, and generally any application where agents are programmed by different parties with competing interests. Characteristic features of such systems are limited trust, agent heterogeneity and unpredictable behaviour. The agents in these systems, although required to make decisions and act locally, operate in the context of well-defined protocols and procedures for interacting with other agents, and agreed norms of behaviour that govern these interactions. Another characteristic of open agent systems is the high probability of non-conformance to these standards. It has been argued that a specification of systems of this type should satisfy at least the following two requirements: first, the interactions of the members should be governed by a formal, declarative, verifiable and meaningful semantics [18]. Second, to cater for the possibility that agent behaviour may deviate from what is prescribed, agent interactions can usefully be described in terms of permissions, obligations and other more complex normative relations that may exist between them [10].

We have been developing a theoretical framework for the executable specification of open agent systems that addresses the aforementioned requirements [1, 2]. We adopt the perspective of an external observer and view agent systems as instances of *normative systems* [10]. We thus represent the institutional powers [11], permissions and obligations of the agents in such systems. In particular, we employ an action formalism to specify the constraints that govern the behaviour of the members and then use a computational framework to animate the specification and investigate its properties. For the action formalism, we have employed both the Event Calculus and the language $C+$ [8, 7, 6], a language with explicit transition system semantics. We have also been developing an extended form of $C+$ specifically designed for modelling the institutional aspects of agent systems [16]. In this paper we will use the language $C+$. An example of our use of the Event Calculus may be found in [1].

We demonstrate how the theoretical and computational frameworks can be applied to specifying and executing an argumentation protocol based on Brewka's reconstruction [3], in the Situation Calculus [14], of a theory of formal disputation originally proposed by Rescher [15]. This protocol is typical of the kind of protocols that are encountered in MAS, and in providing a procedure for the conduct of debate and dispute resolution, it addresses a number of questions of relevance to the field of artificial intelligence and law.

The paper is structured as follows. First, we briefly describe the $C+$ language. Second, we present a theoretical framework for specifying open agent systems, and a computational framework for executing these specifications. Third, we summarise Brewka's reconstruction of Rescher's theory of formal disputation. Fourth, we specify and execute (a slightly modified form of) Brewka's argumentation protocol with the use of the theoretical and computational frameworks. Finally, we discuss related work, summarise, and point out directions for future research.

2. THE $C+$ LANGUAGE

The action language $C+$ [8, 7, 6] enables the representation of the effects (direct and indirect) of actions and default persistence ('inertia') of fluents from state to state. An action

ICAIL'03 June 24-28, 2003, Edinburgh, UK

description in $C+$ is a set of $C+$ laws that define a transition system of a particular kind. The representation of an action domain in $C+$ consists of *rigid* constants, *fluent* constants and *action* constants[1]. Rigid constants are symbols that represent the features of the system whose value is fixed and does not depend on a state. Fluent constants are symbols characterising a state. They are divided into two categories: simple fluent constants and statically determined fluent constants. Simple fluent constants are related to actions by *dynamic laws* (that is, rules describing a transition from a state s_i to its successor state s_{i+1}). Statically determined fluent constants are characterised by *static laws* (that is, rules describing an individual state) relating them to other fluent constants. Static and dynamic laws are defined below. Action constants are symbols characterising state transitions.

An *action signature* is a non-empty set σ^{rf} of rigid and fluent constants and a non-empty set σ^{act} of action constants. An action description D in $C+$ is a set of *causal laws*. A causal law can be either a *static law* or a *dynamic law*. A static law is an expression:

$$\textbf{caused } F \textbf{ if } G \tag{1}$$

where F and G are formulas (that is, truth-functional compounds) of rigid and fluent constants. In a static law constants in F and G are evaluated on the same state. A dynamic law is an expression:

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{2}$$

where F, G and H are formulas such that every constant occurring in F is a simple fluent constant, every constant occurring in G is rigid or fluent, and H is any combination of rigid constants, fluent constants and action constants. In a transition from state s_i to state s_{i+1} constants in F and in G are evaluated on s_{i+1}, fluent and rigid constants in H are evaluated on s_i and action constants in H are evaluated on the transition itself. F is called the *head* of the static law (1) and the dynamic law (2).

The $C+$ language provides various abbreviations for causal laws. These will be used extensively in this paper. For example, the *closed-world assumption* regarding a fluent F, whereby F is assumed to hold in the absence of information to the contrary, is represented as:

$$\textbf{default } F$$

The above expression is an abbreviation for the static law:

$$\textbf{caused } F \textbf{ if } F$$

A dynamic law of the form:

$$\textbf{caused } F \textbf{ if } \top \textbf{ after } H \wedge a$$

is often abbreviated as (here we separated the action constant a from H):

$$a \textbf{ causes } F \textbf{ if } H$$

In the case where H is $\top$ the above abbreviation is expressed as follows:

$$a \textbf{ causes } F$$

[1]We are describing here the version of $C+$ presented in [8, 7]. A very recent presentation [6] contains some extensions to the form of rules in $C+$, but nothing that materially affects what we are describing in this paper.

The persistence ('inertia') of a fluent constant c over time is represented as:

$$\textbf{inertial } c$$

This is an abbreviation for the *set* of dynamic laws of the form (for all values u of the c fluent constant):

$$\textbf{caused } c = u \textbf{ if } c = u \textbf{ after } c = u$$

It is not possible in the space available here to give a full account of the $C+$ language and its semantics, and to explain in detail how this provides a treatment of defaults and persistence ('inertia'). We trust that the $C+$ language, and especially its abbreviations, are sufficiently natural that readers can follow the presentation of the example in later sections. Interested readers are referred to [7, 6] for technical details. For completeness, we give here a (rather concise) statement of the semantics.

Given an action description D, a *state* is an interpretation of σ^{rf} that satisfies $G \rightarrow F$ (G materially implies F) for every static law (1). An *action* is a propositional interpretation of σ^{act}. A *transition* is a triple (s, a, s') where s is the initial state, s' is the resulting state and a is an action. A formula F is *caused* in a transition (s, a, s') if $F \in T_{static}(s') \cup E(s, a, s')$, where:

1. $T_{static}(s) = \{F \mid \text{static law } (1) \text{ is in } D, s \models G\}$
2. $E(s, a, s') = \{F \mid \text{dynamic law } (2) \text{ is in } D, s' \models G, s \cup a \models H\}$.

A transition (s, a, s') is *causally explained* by D iff:

1. $s' \models T_{static}(s')$ and $s' \models E(s, a, s')$.
2. There is no other s'' such that $s'' \models T_{static}(s')$ and $s'' \models E(s, a, s')$.

The transition system described by a $C+$ action description D is a directed graph where: the vertices are the states of D, and an edge from state s to state s' is labelled with action a if the transition (s, a, s') is causally explained by D.

3. A THEORETICAL FRAMEWORK

In previous work we presented a theoretical framework for specifying open agent systems [1, 2]. We classify an agent system as 'open' if it satisfies the following properties. First, there is no direct access to an agent's internal ('mental') state. Second, the members of the system do not necessarily have a notion of global utility. Members may fail to, or even choose not to, conform to the norms governing their behaviour in order to achieve their individual goals. In addition to these properties, in open agent systems the behaviour of the members cannot be predicted in advance [9]. Next, we briefly describe the theoretical framework for the specification of open agent systems.

We have employed both the Event Calculus and the $C+$ language to formalise open agent systems; each has its advantages and disadvantages. In this paper we will use $C+$. We focus on the specification of the *social constraints*, *social roles* and *social states* of open agent systems. The social constraints dictate the behaviour of the members of open agent systems by specifying their institutional powers [11], normative positions (permissions and obligations), and sanctions. These constraints are expressed by means of a $C+$ action description, say D^{soc}. The social roles are associated with a set of preconditions that agents must satisfy in order to be eligible to occupy these roles, and with a set of (role) constraints that govern the behaviour of the agents that occupy particular roles within the system. The role

constraints and the constraints that describe the preconditions of the roles are a subset of the set of social constraints. The $C+$ action description D^{soc} defines a transition system whose states we refer to as 'social states'. They are interpretations of the set of constants σ^{rf} of D^{soc}, which includes fluent constants representing the powers, normative positions and sanctions associated with each member.

Our specification is based on four assumptions. First, we specify the constraints at the design stage of the systems. Furthermore, we assume that the specification of the social constraints does not change during the execution of the systems; or if it does change, that these changes can be localised to changes in values of various parameters. Second, our specification is based only on externally observable states of affairs and actions and not on the internal architecture of the individual agents. Furthermore, the specification of the social constraints refers to the externally observable behaviour of the agents and not to the way agents reason about their behaviour. (This is in contrast to some other approaches in MAS where the semantics of agent communication is expressed is terms of the agents' internal, mentalistic states, such as their beliefs, desires, and intentions.) Third, we limit attention to a single computational system, even though we anticipate applications where agents are members of several different systems simultaneously. Fourth, apart from its members, an open agent system may include other groupings, which we call, following standard usage, *institutions* [11, 5]. Such institutions have their own constraints, roles, communication language, and so on. However, for simplicity we again restrict attention to open agent systems that can be modelled as single institutions. Due to the last two assumptions, in the following sections we do not relativise the specification of social constraints to a particular system or institution. The relaxation of these single-institution assumptions raises a number of further complications and will be presented elsewhere.

3.1 Social Constraints

Social constraints are represented as a set of $C+$ static and dynamic laws. The social constraints define, amongst other things, the semantics of the agents' actions, expressed by means of dynamic laws which specify what states of affairs (combinations of fluent constants) each action initiates and terminates. In different systems, or in different institutions within the same system, the same action might have different semantics, in that it initiates or terminates different states of affairs (combinations of fluent constants).

We maintain the standard and long established distinction between *permission*, *physical capability* and *institutionalised power* [11]. Accordingly, we have three levels of specification of the social constraints of open agent systems. We describe these three levels next.

3.1.1 Institutionalised power and valid actions

The term institutional (or 'institutionalised') power refers to the characteristic feature of organisations—legal, formal, or informal—whereby designated agents, often when acting in specific roles, are empowered to create or modify states of affairs of special significance in that organisation, usually by performing a specified kind of act (such as when a priest performs a marriage, or an agent signs a contract, or the chairperson of a formal meeting declares the meeting closed). This concept has received considerable attention within the jurisprudential literature, usually under the headings of 'legal power', 'legal capacity' or 'norms of competence', but it is clear that it is not an exclusively *legal* phenomenon—it is a standard feature of all organised interaction. The term 'institution' has been coined as a suitably neutral term, to encompass legal systems, formal organisations, and informal groupings.

According to the account given in [11], institutional power can be seen as a special case of a more general phenomenon whereby an action, or a state of affairs, A—because of the rules and conventions of an institution—counts, in that institution, as an action or state of affairs B (as when sending a letter with a particular form of words counts as making an offer, or raising a hand counts as making a bid at an auction, or banging the table with a wooden mallet counts as declaring a meeting closed).

For the specification of the effects of actions within institutions, it is important—essential—to distinguish between, for example, the act of making an offer and the act by means of which that offer is made (such as sending a letter). Banging the table with a wooden mallet is not the same act as closing a meeting. Indeed, it is only if the table is banged by a person with the institutional power to close the meeting that the meeting is thereby declared closed; the same act performed by an agent without this power has no effect on the status of the meeting (though it may have other effects). In such examples we say that an agent 'has the institutional power' (or just 'power'), or 'is empowered', to close the meeting by means of banging the table with a wooden mallet.

In some circumstances it is awkward or unnecessary to isolate and name all instances of the acts by means of which agents exercise their institutional powers. When describing an auction, for example, it is convenient to say 'agent x made a bid' and let context disambiguate whether we mean by this that the agent performed an action, such as raising its hand, by means of which the making of a bid is signalled, or whether agent x actually made a bid, in the sense that the current bidding price of the item under auction was changed. We find it convenient to disambiguate in these circumstances by attaching the label 'valid' to act descriptions. We say that an action is *valid* at a point in time if and only if the agent that performed that action had the *institutional power* to perform it at that point in time. So, when we say that 'agent x made a bid y' we mean, by convention, merely that agent x signalled its intention to make a bid y; this act was not necessarily effective in changing the current bidding price. In order to say that the bidding price was actually changed, we say that the action 'agent x made a bid y' was valid: not only did x signal its intention to make bid y, but also x was empowered to make the bid y at that time. Similarly, 'invalid' is used to indicate lack of institutional power: when we say 'agent x made a bid y' is invalid we mean that x signalled its intention to make bid y but did not have the institutional power to make that bid at that time (and so the attempt to change the current bidding price was not successful). Differentiating between valid ('meaningful') and invalid ('meaningless') actions is of great importance in the analysis of agent systems. In an auction, for example, the auctioneer has to determine which bids are *valid*, and therefore which bids are *eligible* for winning the auction.

We are conscious that this use of the term 'valid' is not ideal and indeed may be inappropriate in some contexts.

Terms such as 'valid', 'in order', 'proper' (and 'invalid', 'out of order', 'improper', 'void') have specific meanings in certain contexts. However, these contexts are relatively few, and the same meaning is not always given in each. It is difficult to find a suitably neutral term—we will stick to the term 'valid' in this paper.

In the specification of social constraints in open agent systems, the definition of institutional powers is application-specific. In this paper, the definitions of powers and valid actions are formulated as static and dynamic laws of a $C+$ action description.

3.1.2 Permitted Actions

The second level of specification of the social constraints provides the definition of *permitted*, *prohibited* and *obligatory* actions. These definitions are application-specific. In some cases, we might want to associate powers with permissions. For example, in some systems, an agent is permitted to perform an action if that agent is empowered to perform that action. In other systems the relationship is stronger: an agent is permitted to perform an action if *and only if* it is empowered to perform that action. In general, however, there is no standard, fixed relationship between powers and permissions. For example, it is sometimes valuable to *forbid* an agent to perform an action even if it is empowered to perform that action. Similarly, the specification of obligations is application-specific. However, it is important to maintain consistency of the specification of permissions and obligations on the same system: an agent cannot be forbidden and obliged to perform the same action at the same time.

Determining what actions are permitted, prohibited or obligatory enables the classification of the behaviour of individual agents and the system as a whole into categories such as 'social' or 'anti-social', 'acceptable' or 'unacceptable', and so on. For example, the behaviour of an agent might be considered 'anti-social' or 'unacceptable' if that agent performs certain forbidden actions or does not comply with its obligations. Based on the behaviour of the individual agents, it is possible to classify the behaviour of the system as a whole. For example, the state of a system may be considered 'unacceptable' if the majority of its members have not complied with their obligations.

3.1.3 Enforcement Policies

At the third level of specification of the social constraints there is the definition of *sanctions* and *enforcement policies*. Sanctions are a means of dealing with 'anti-social' or 'unacceptable' behaviour. This level of specification has two aspects: (i) when is an agent sanctioned, and (ii) what is the penalty that the agent has to face (in the case that it does get sanctioned). The specification of both of these aspects is also application-specific. As far as the first is concerned, agents may be sanctioned for not complying with their obligations, or they may be sanctioned if they perform forbidden actions. As regards the second aspect, penalties can come in many different forms. The house rules of an auction house, for example, may stipulate that bidders who bid out of turn (and are therefore considered 'sanctioned') are no longer empowered to enter other auctions. In different settings, the same type of misbehaviour might create different sanctions. One common type of sanction may be expressed in terms of a social concept such as *bad reputation*.

Sanctions are one means by which an open agent system may discourage 'unacceptable' or 'anti-social' behaviour. Another mechanism is to try to devise additional controls (physical or institutional) that will force agents to comply with their obligations or prevent them from performing forbidden actions. In an automated auction, for example, forbidden (non-permitted) bids may be physically blocked, in the sense that their transmission is disabled, or the specification of a valid bid may be changed to render them ineffective. The general strategy of designing mechanisms to force compliance and eliminate non-permitted behaviour is what Jones and Sergot [10] referred to as *regimentation*. Sanctioning mechanisms are required because the opportunities for effective regimentation are usually very limited.

4. A COMPUTATIONAL FRAMEWORK

This section describes a computational framework for executing the specifications of open agent systems. The computational framework makes use of a software implementation, the Causal Calculator (CCALC), a system designed and implemented by the Action Group of the University of Texas for performing computational tasks on $C+$ action descriptions [6, 7]. CCALC has two inputs: a $C+$ action description D and a *query* concerning D.

CCALC supports the computation of prediction, planning, and 'postdiction' queries. We use CCALC as follows: in the computation of each query, CCALC has as input the specification of the social constraints expressed as a $C+$ action description. Given as additional input a *narrative*, a description of temporally-ordered externally observable events, the computation of prediction queries produces the social state of an agent system, that is, a representation of the powers, normative positions, sanctions, and roles associated with each member of the computational system at each timepoint. The computation of planning queries produces sequences of actions that lead from a given initial state to a given final state. The computation of this type of query helps to *validate* (prove properties of) the specification of the social constraints. In [2], for example, we validated the specification of a contract-net protocol [17] using CCALC's computation of planning queries. The computation of 'postdiction' queries—in which the task is to explain how a (partially described) social state could have arisen—can also be useful in validating the specification of the social constraints.

These tasks (queries) are intended to be performed both at design time, or 'off-line', and at run-time, or 'on-line' (in the terminology of [12]). The on-line services can be provided by a central server or in various distributed configurations. The off-line and on-line services as well as the on-line configurations, including the adequacy of the CCALC implementation for supporting on-line services, will be discussed in a forthcoming paper.

5. A THEORY OF FORMAL DISPUTATION

In order to illustrate the utility of the theoretical and computational frameworks, in this section we describe, specify and animate an argumentation (disputation) protocol based on Brewka's reconstruction [3] of Rescher's Theory of Formal Disputation (RTFD) [15]. We have picked this example because (a) in defining a set repertoire of possible moves for each participant, and their effects, it is typical of the kind of protocols that are encountered in MAS, (b) in providing a procedure for the resolution of a (simple kind of) dispute,

it addresses a number of other questions of potential relevance to the field of artificial intelligence and law, and (c) Brewka's formalisation provides a natural starting point.

According to RTFD, an argumentation process may be viewed as a three-player game: the *proponent* claims a particular thesis and the *opponent* may question this thesis. The *determiner* decides whether the proponent's thesis was successfully defended or not. Due to space limitations we do not describe Rescher's original theory of disputation but only elements of Brewka's reconstruction and our variation of it.

An argumentation system, according to Brewka's definition (see [3, Definition 4.9]), includes as core components a *logic of disputation* and an *argumentation context*. In Brewka's reconstruction, the logic of disputation is preferential default logic [4] and the argumentation context is formalised with the use of (a dialect of) the Situation Calculus [14]. The main actions of the protocol are the following: *claiming*, *conceding to*, *retracting*, and *denying* propositions and default rules, *declaring* the winner of the argumentation, and *objecting* to actions performed by the other participants.

The semantics of the protocol actions are given in terms of the *premises* held by the proponent and opponent. The fluent $premise(ag, q, s)$ expresses that "in situation s agent ag is committed to accepting q, where q is a formula in the logic of disputation" [3, pp.266–267]. The premises of agent ag are the formulas that ag holds explicitly. The related fluent $accepts(ag, q, s)$ is used to represent the formulas that ag holds implicitly: $accepts(ag, q, s)$ expresses that q follows in the logic of disputation L from the premises explicitly held by agent ag in argumentation record s:

$$accepts(ag, q, s) \text{ iff } \{p \mid premise(ag, p, s)\} \vdash_L q \qquad (3)$$

An argumentation record is a situation (in the terminology of the Situation Calculus) and so includes the history of the protocol.

The semantics of a *claim* action, for example, of a proposition or a default rule, are given by the following Situation Calculus *effect* axiom:

$$premise(ag, q, do(claim(ag, q), s)) \qquad (4)$$

Expression (4) states that the successor situation following the performance of a claim action by ag includes a premise about the claimed proposition (or default). In expression (4) q represents either a proposition or a default rule of the form $n :: a : b/c$ where n is a label associated with the default rule, a is the prerequisite, b is the justification and c is the consequent of the rule [3].

Brewka distinguishes between 'possible' and 'legal' actions. Possible actions are specified by means of Situation Calculus *possibility* axioms. Consider the following possibility axiom of the retract action:

$$poss(retract(ag, q), s) \leftrightarrow premise(ag, q, s) \qquad (5)$$

The above axiom states that it is 'possible' for an agent to retract a proposition (or a default) q if and only if that agent has accepted that proposition (or default) as a premise. The conditions that determine whether an action is possible or not are specified in a protocol-independent manner—as usual in Situation Calculus, possibility axioms are used to specify the well-formed actions of the formalisation.

'Legal' actions, in contrast to possible actions, are specified in a protocol-dependent manner. Consider the following example of a legal action:

$$legal(declare(det, pro), s) \rightarrow accepts(pro, topic, s) \qquad (6)$$

The above expression states that declaring the proponent 'pro' as the winner is legal only if the proponent accepts the topic of the argumentation. Legal actions, in the terminology adopted in this paper, may be viewed in the first instance as 'valid' actions, though it may be that what Brewka calls 'legal' is intended to cover also what we would call 'permitted'. Elaboration of this point is one of the main aims of the formalisation presented in later sections.

A point of departure of Brewka's reconstruction from the original theory is the introduction of the 'object' action. The participants of argumentation protocols may perform invalid actions; the effects of an invalid action are the same as if the action were a valid one—provided that no other participant objects to the invalid action. If some participant objects immediately (that is, no other action takes place between the invalid action and the objection), then the effects of the invalid action are 'cancelled'.

The 'object' mechanism is not a new idea in the field of argumentation protocols. Prakken [13] points out that an 'object' mechanism of this type is part of Robert's Rules of Order (RRO): "[t]he general rule is that anything goes until any member objects, after which RRO must be strictly applied" [13, p.10]. One can find similar mechanisms in most procedures for the conduct of formal debates and disputes.

Enabling agents to object to invalid actions can lead to a more flexible argumentation protocol. In Brewka's modification of RTFD, for example, the proponent might choose not to object to an invalid action performed by the determiner because it (the proponent) calculates that the invalid action will serve its benefit better than having the invalid action ruled out. However, it can be argued that Brewka's object mechanism is too simplistic, if it is a model of how argumentation processes are actually conducted, and too rigid, if it is a model of how argumentation processes ought to be conducted. Consider for example the case where an agent, say the determiner, repeatedly performs invalid actions. The proponent and opponent have to object to every invalid action performed by the determiner because if they do not object, they implicitly accept the invalid actions as valid ones. Moreover, according to Brewka's treatment, an objection will 'undo' the effects of an invalid action if and only if the objection takes place immediately after the invalid action. If for some reason an agent is not (physically) capable of immediately objecting to an invalid action then it will be considered that this agent does not object and so implicitly agrees to the treatment of the invalid action as a valid one. Prakken [13], in his formalisation of RRO, discusses the possibility of 'undoing' the effects of an invalid action by appealing later to an adjudicating authority, in the case where it was not physically possible to object immediately to the invalid action.

Finally, Brewka's reconstruction formalises Rescher's 'silence implies consent' principle. This principle may be summarised as follows: an agent, say the proponent, is assumed to accept a proposition as a premise, if the opponent accepts this proposition as a premise, and as long as it (the proponent) does not deny or retract that proposition.

5.1 A Variation of Brewka's RTFD

Although it is our general aim in this paper to present a

reconstruction of Brewka's account of RTFD we make two adjustments to his (Brewka's) version. First, we further refine the distinction between 'possible' and 'legal' actions. Brewka does not discuss the setting within which his formalisation will be employed. We have in mind a setting where autonomous software agents in a MAS engage in the argumentation as part of a negotiation or dispute resolution process. We therefore have to consider not only what kinds of actions are well-formed (one possible interpretation of the Situation Calculus predicate *poss*—see, for example, axiom (5)) but also which of these actions each agent will be *practically able* to perform at each stage of a given implementation.

Second, although Brewka states that the argumentation regarding a proposition may terminate due to a deadline, he provides no further details about how this would work and how such deadlines would affect the argumentation process. There is also the question of the 'silence implies consent' principle and how that would fit in the practical setting.

Here then is our variant of Brewka's reconstruction of RTFD. We will refer to it as RTFD*. Initially, only the proponent has the institutional power to claim a proposition. This proposition is the topic of the argumentation. The argumentation commences when the proponent makes its claim—any other action does not count as the commencement of the protocol. The proponent and opponent then take it in turn to perform actions—in the setting we have in mind, these actions would be transmissions of messages to the other participants, to claim, deny, retract or concede to a proposition (or a default rule), or to object to an action by one of the other participants. During the process, the proponent and opponent must perform their chosen actions by specified deadlines (timeouts). Without this feature there is no practical way of controlling the exchanges, of determining whether a participant has 'spoken', because otherwise one might have to wait indefinitely for messages to arrive over the communication channels. For similar reasons, it is also necessary to impose a limit on the number of exchanges, or on the total elapsed time for the argumentation process. The determiner is empowered to declare the winner only at the end of the argumentation, that is, when the specified time period for the argumentation elapses. If at that time both the proponent and opponent have accepted the topic of the argumentation, then the determiner is only empowered to declare the proponent as the winner. If, however, the proponent does not accept the topic at that time then the determiner is only empowered to declare the opponent as the winner. Finally, if the proponent accepts the topic and the opponent does not, the determiner is empowered to declare either the proponent or the opponent as the winner. In this case, we say that the determiner has *discretion* to declare either of them as the winner. We will comment separately on the 'silence implies consent' principle.

The fact that we depart in some respects from Brewka's reconstruction means of course that we cannot provide a formal comparison between Brewka's formalisation in the Situation Calculus and our formalisation in the $C+$ language—for example, to prove an equivalence of some sort between them. Informally, our formalisation differs in the following respects. First, the proponent, opponent and determiner are empowered to perform actions in specified time periods. Such a specification is closer to 'realistic' argumentation protocols to be executed by software entities (or human agents communicating remotely over a communication channel). Second, in addition to distinguishing between (practically) possible actions and institutional powers, we specify the permissions, obligations and sanctions associated with the participants of the argumentation protocols. Finally, even though we depart from Brewka's reconstruction in the aforementioned points, our version maintains the feature of (immediately) objecting to invalid actions.

Table 1: A Partial $C+$ Description of RTFD*.

Notation:
$roleName$ ranges over $\{Proponent, Opponent, Determiner\}$
$protag, protag'$ ranges over $\{Pro, Opp\}$
ag, ag' range over $\{Pro, Opp, Det\}$
p, q range over a finite set of propositions
act ranges over $\{Claim(q), Concede(q), Retract(q),$ $Deny(q), Object(ag{:}act), Declare(protag)\}$
Rigid constants:
$Role\_of(ag)$=$roleName$,
$Topic$=$p, Implies(p, q)$ (both boolean)
Simple fluent constants:
$Turn$=$roleName$,
$Premise(protag, q), Sanctioned(ag)$ (both boolean),
Statically determined fluent constants (boolean):
$Pow(ag, act), Permitted(ag, act),$ $Obliged(ag, act), Accepts(protag, q)$
Causal laws:
inertial c for every simple fluent constant c

5.2 Specification of RTFD* in $C+$

We present an action description D^{RTFD^*} that expresses the specification of the social constraints of an argumentation protocol RTFD*. Table 1 shows a subset of the causal laws and a subset of the action signature of D^{RTFD^*}. We will call 'pro' the agent that occupies the role of the proponent, 'opp' the agent that occupies the role of the opponent and 'det' the agent that occupies the role of the determiner.

In $C+$ predicates and constants start with an upper-case letter and variables start with a lower-case letter. Variables are assumed to be universally quantified unless otherwise indicated. In the following analysis, terms of the form $ag{:}act$ are used as action symbols representing the performance of action act by agent ag. This is valid $C+$ syntax but, because of restrictions in the input language of CCALC we have to modify this representation slightly for input to the CCALC implementation. This is a minor point of detail and does not affect the semantics of D^{RTFD^*}.

5.3 Social Constraints

5.3.1 Institutionalised power and valid actions

According to our specification of RTFD*, the proponent and opponent are empowered to perform actions in turn, and the determiner is empowered to declare the winner, in specified circumstances, when the exchange is complete. Timeout events initiate and terminate the powers of the participants of the argumentation; we will discuss their representation presently.

The actions that the participants may perform are classified as 'valid' as described in section 3.1.1—that is, an

action is valid if and only if the agent that performs the action has the institutional power to do so. We have chosen not to include a constant 'valid' in the action signature of D^{RTFD^*}, however. Valid actions are represented implicitly; institutional powers are represented explicitly.

We represent the institutional powers with the use of statically determined fluents of the form $Pow(ag, act)$ defined by means of static laws. For example, the power of an agent to retract a proposition (or a default rule) is defined as follows:

$$\begin{aligned}&\textbf{caused } Pow(protag, Retract(q)) \textbf{ if } Active \wedge \\ &\quad (Turn{=}Role\_of(protag)) \wedge Premise(protag, q)\end{aligned} \tag{7}$$

Here *protag* is a variable that ranges over the two protagonists, proponent and opponent, and q is a variable that ranges over the formulas (propositions and default rules) that the two protagonists may claim, retract, deny, and so on. We assume that a finite number of such formulas can be identified and specified at the outset. This is necessary for implementation in $C+$ and the CCALC system, though not necessary in other formalisms.

Constraint (7) states that an agent *protag* is empowered to retract a proposition (or a default rule) q when: (i) the protocol is currently active, (ii) it is the agent *protag*'s turn to 'speak' (values of the *Turn* fluent constant are role names—see Table 1), and (iii) the agent *protag* currently accepts q as one of its (explicit) premises. The *Active* simple fluent constant is initially false and becomes true when the proponent performs the first valid claim about the topic of the argumentation, that is, when the protocol commences.

The closed-world statement that institutional powers do not hold unless defined by a suitable static law is expressed by a constraint of the following form:

$$\textbf{default } \neg Pow(ag, act) \tag{8}$$

Here ag is a variable that ranges over all three participants, proponent, opponent, and determiner, and *act* is a variable that ranges over all the acts that can be performed by any of the participant agents during the argumentation.

The question of whether an agent is *permitted* to 'speak' when it is not currently empowered to speak, or whether the implementation makes it practically *possible* for the agent to speak when it is not empowered to speak are separate elements of the system specification. They will be considered in separate sub-sections later.

It is a feature of Brewka's reconstruction of RTFD that agents may object to other agents' actions. So suppose that agent (proponent or opponent) ag has retracted q and now it is the turn of agent ag' to 'speak'. One of the moves available to ag' is to object to ag's retraction of q. This move is always available but the objection is only effective in undoing the effects of the retraction when ag' is *empowered* to object. Again, the question of whether ag' is permitted to object or practically able to object when not empowered to do so is a separate question to be considered at other levels of the specification.

When is agent ag' empowered to object to an action *act* performed by an agent ag? When the last action in the protocol was ag's performance of *act* but ag was not empowered to perform *act* at that time. There are several ways to express this in $C+$, of which the simplest is as follows:

$$\begin{aligned}&ag{:}act \textbf{ causes } Pow(ag', Object(ag{:}act)) \textbf{ if} \\ &\quad Active \wedge (ag \neq ag') \wedge \neg Pow(ag, act)\end{aligned} \tag{9}$$

Here ag and ag' are variables ranging over the three participants. Notice that the formulation (9) allows for objections, objections to objections, objections to objections to objections, and so on. There is no particular difficulty in accommodating nested objections of this kind if we wish but we can also follow Brewka and eliminate them by replacing the general rule schema (9) by the relevant instances.

Strictly speaking, formulation (9) does not conform to the syntax of the $C+$ language because a statically determined fluent constant, $Pow(\dots)$, appears in the head of a dynamic law, that is, law (9). In the actual implementation of the protocol (see Section 5.5) we use inertial simple fluent constants to represent the power to object and statically determined fluent constants to represent the power to perform any other action. This is another minor point of detail that does not affect the semantics of D^{RTFD^*}.

In the formulation presented so far, an agent ag' is empowered to object to a retraction of q by ag either if ag has just spoken out of turn or if ag has retracted a formula q that does not hold as a premise. We can represent various other alternatives. For example:

$$\begin{aligned}&ag{:}Retract(q) \textbf{ causes } Pow(ag', Object(ag{:}Retract(q))) \textbf{ if} \\ &\quad Active \wedge (ag \neq ag') \wedge \\ &\quad Premise(ag, q) \wedge \neg Pow(ag, Retract(q))\end{aligned} \tag{10}$$

In more complicated examples it may be useful to break down the classification of 'valid' actions into smaller components, distinguishing for example between well-formed (or 'proper') actions and not well-formed (or 'improper') actions. For instance, the retraction of a premise q by an agent ag could be said to be well-formed if and only if q is one of the premises of agent q. We are currently trying to devise a more refined classification along these lines.

An interesting feature of the protocol is that the retraction of a premise q by an agent can be effective even if that agent is not empowered to retract q. So we have (for *protag* either proponent or opponent):

$$protag{:}Retract(q) \textbf{ causes } \neg Premise(protag, q) \tag{11}$$

In other words, a retraction by either of the protagonists always has an effect. It is just that a subsequent objection by another participant may undo the effect of the retraction. It will undo the retraction if the objection is made by an agent empowered to object, but not otherwise.

Objecting to an invalid action (whilst having the power to do so) 'cancels' the effects of the invalid action. In the example, a valid objection re-instates the previously retracted premise:

$$\begin{aligned}&ag{:}Object(protag{:}Retract(q)) \textbf{ causes } Premise(protag, q) \textbf{ if} \\ &\quad Pow(ag, Object(protag{:}Retract(q)))\end{aligned} \tag{12}$$

This is one possible formulation. There are other ways of looking at it—for instance, one may prefer to say that all agents have the power to retract (at least, to make well-formed retractions) whether it is their turn to speak or not, but in certain circumstances, including speaking out of turn, that retraction will create a power of the opposing agent to object, and thereby to undo the retraction.

Which is the better formulation? Both can be expressed in the framework, and to a large extent it is a matter of taste.

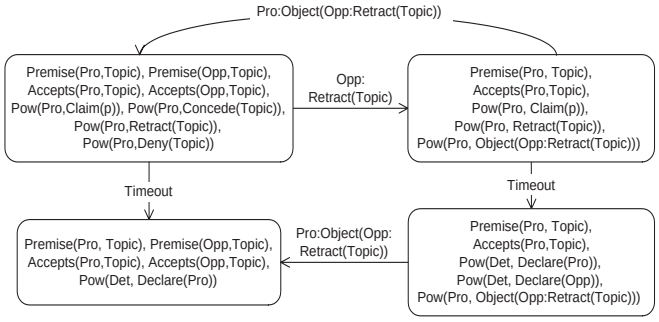

Figure 1: Objecting to an Invalid Retraction.

In this example, we might guess that the alternatives are equivalent, in the sense that they produce protocols that always have the same outcome. This is a hypothesis that can be tested. One aim of the work presented here is to provide computational tools to support the automated testing of such hypotheses.

The other actions of the protocol—claiming, conceding and denying propositions and default rules, and declaring the winner—can be formulated in similar fashion. Deadlines can be treated by means of *Timeout* action constants which simply switch the value of the *Turn* fluent.

Figure 1 shows a fragment of the transition system defined by D^{RTFD^*}. It is worth noting, in this example, the difference between objecting and not objecting to an invalid retraction. In the first case, the sequences of actions *Retract*, *Object*, *Timeout* and *Retract*, *Timeout*, *Object* lead to a state where the determiner is only empowered to declare the proponent as the winner (because both protagonists accept the topic of the argumentation) whereas in the second case, the sequence of actions *Retract*, *Timeout* leads to a state where the determiner is empowered to declare either the proponent or the opponent as the winner (because only the proponent accepts the topic of the argumentation). In this example, the timeout event signals the end of the disputation and empowers the determiner to declare the winner.

5.3.2 Permitted Actions

For the sake of a simple example for illustrative purposes, let us suppose that an argumentation protocol forbids the proponent and opponent from retracting a proposition (or a default) even when they are empowered to do so. Apart from this, each agent is permitted to perform any action it is empowered to perform:

$$\textbf{caused } \mathit{Permitted}(ag, act) \textbf{ if } \\ \mathit{Pow}(ag, act) \wedge (act \neq \mathit{Retract}(q)) \tag{13}$$

$$\textbf{default } \neg \mathit{Permitted}(ag, act) \tag{14}$$

Constraint (13) states that retracting propositions (or defaults) is considered 'unacceptable' according to this specification of RTFD*.

Obligations can also be associated with any of the agents, though in this simple example we associate them only with the determiner. At the end of the disputation (as signalled by a timeout), if both protagonists have accepted the topic of the argumentation, the determiner is only empowered to declare the proponent as the winner, and is obliged to do so:

$$\textbf{caused } \mathit{Obliged}(\mathit{Det}, \mathit{Declare}(\mathit{Pro})) \textbf{ if } \\ \neg \mathit{FinalState} \wedge \mathit{Accepts}(\mathit{Pro}, \mathit{Topic}) \wedge \\ \mathit{Accepts}(\mathit{Opp}, \mathit{Topic}) \wedge (\mathit{Turn}{=}\mathit{Determiner}) \tag{15}$$

The *FinalState* simple fluent constant becomes true either when the determiner declares the winner of the argumentation (having the power to do so) or when the last timeout takes place thus signalling the end of the protocol.

Should it happen that at the end of the argumentation the proponent no longer accepts the topic of the argumentation, the determiner is only empowered to declare the opponent as the winner, and moreover is obliged to do so:

$$\textbf{caused } \mathit{Obliged}(\mathit{Det}, \mathit{Declare}(\mathit{Opp})) \textbf{ if } \\ \neg \mathit{FinalState} \wedge \neg \mathit{Accepts}(\mathit{Pro}, \mathit{Topic}) \wedge \\ (\mathit{Turn}{=}\mathit{Determiner}) \tag{16}$$

The closed-world statement about obligations is expressed by a constraint of the following form:

$$\textbf{default } \neg \mathit{Obliged}(ag, act) \tag{17}$$

There are a number of additional cases where the determiner is empowered to declare the winner either way, though there may be an obligation on it to declare one way rather than the other. We describe these next.

5.3.3 Enforcement Policies

Since (for the sake of an example) we are supposing that retractions, valid or not, are never permitted, a protagonist is sanctioned whenever it retracts a proposition (or a default):

$$\mathit{protag}{:}\mathit{Retract}(q) \textbf{ causes } \mathit{Sanctioned}(\mathit{protag}) \textbf{ if } \\ \neg \mathit{Permitted}(\mathit{protag}, \mathit{Retract}(q)) \tag{18}$$

The penalty that a sanctioned proponent or opponent has to face (for the sake of the example) is given in terms of the determiner's obligations. If at the close of the disputation the proponent still accepts the topic but the opponent does not, the determiner is obliged to declare as the winner the proponent, provided that the proponent did not perform any non-permitted actions (and the opponent did perform non-permitted actions). In other words, the penalty the opponent pays for its non-permitted actions is that it will not win an unresolved dispute if the proponent does not perform forbidden actions (and the determiner complies with its obligations):

$$\textbf{caused } \mathit{Obliged}(\mathit{Det}, \mathit{Declare}(\mathit{Pro})) \textbf{ if } \\ \neg \mathit{FinalState} \wedge (\mathit{Turn}{=}\mathit{Determiner}) \wedge \\ \mathit{Accepts}(\mathit{Pro}, \mathit{Topic}) \wedge \neg \mathit{Accepts}(\mathit{Opp}, \mathit{Topic}) \wedge \\ \neg \mathit{Sanctioned}(\mathit{Pro}) \wedge \mathit{Sanctioned}(\mathit{Opp}) \tag{19}$$

A similar obligation can be imposed on the determiner to deal with the case where the proponent is sanctioned and the opponent is not. In the case where both proponent and opponent are sanctioned, the determiner is empowered to declare either of them as the winner but is not obliged to declare one way rather than the other.

What if the determiner fails to fulfil *its* obligations? What sanctions are then enforced? There are a number of possibilities. For example, we might specify that in such a case the determiner is disqualified from acting as a determiner in future argumentations, for a specified time period perhaps.

5.3.4 Additional Considerations

This section briefly describes our formalisation of the acceptance of a proposition and of the 'silence implies consent' principle. Fluent constants of the form $Premise(ag, q)$ represent that q is one of agent ag's explicitly held premises, while fluent constants of the form $Accepts(ag, q)$ are used to represent that q follows, in the logic of disputation, from the explicit premises held by agent ag. For the purposes of the simple experiments we describe in later sections, it is easy to code up the relevant fragments of the logic of disputation as $C+$ rules that statically determine the *Accepts* constants. For example, the following constraint states that an agent accepts all (classical) logical implications of each of its premises.

$$\textbf{caused } Accepts(protag, q) \textbf{ if } \\ Premise(protag, p) \wedge Implies(p, q) \tag{20}$$

The *Implies* here are simply suitably chosen rigid constants. In a similar manner, we specify the acceptance of propositions as a result of the conjunctions of an agent's premises and of premises regarding default rules.

Clearly, this method works only for simple examples and is not a general solution to the incorporation of a logic of disputation in the specification of argumentation protocols. Is there a general solution? We take it that the choice of prioritised default logic is not an essential feature of Brewka's reconstruction, and could just as well be replaced by another formalism with similar properties. The $C+$ language is based on a general nonmonotonic reasoning formalism ('nonmonotonic causal theories' [6]). So one general solution is to replace default rules expressed as formulas of (prioritised) default logic by default rules expressed as formulas of 'nonmonotonic causal theories'. These in turn can trivially be coded as static rules of the language $C+$. But what if a specific default reasoning mechanism *is* required to be used as the logic of disputation in an argumentation protocol? We do not know of any method by which this could be incorporated in $C+$, or in the CCALC implementation.

In order to incorporate the 'silence implies consent' principle in our specification of RTFD*, we need to keep track of the actions of the agents. More precisely, it is necessary to record all the retract and deny actions performed by the proponent and opponent. This can be done by introducing new fluent constants: *ActionHappened*, that record the retract and deny actions and *S_Premise*, that express a premise that has been created due to the 'silence implies consent' principle. The *S_Premise* constants are statically determined as follows:

$$\textbf{caused } S\_Premise(protag, q) \textbf{ if } \\ Premise(protag', q) \wedge (protag \neq protag') \wedge \\ \neg ActionHappened(protag, Deny(q)) \wedge \\ \neg ActionHappened(protag, Retract(q)) \tag{21}$$

The definition of the *Accepts* constants (see, for example, constraint (20)) needs to be (trivially) modified in order to take into account the premises created due to the 'silence implies consent' principle.

Notice that using fluent constants to represent even a partial history of a system's execution and incorporating this as part of a state description runs counter to the spirit of transition systems (and it increases the computational complexity of queries on this transition system very considerably). Generally, there are limits on what can be formulated using transition systems alone, without reference to execution paths (histories). However, further discussion of alternative representation schemes is beyond the scope of this paper.

5.4 Social States

The $C+$ action description D^{RTFD^*} defines a transition system. We may prove various properties of this transition system. For example:

Proposition 1. The transition system that is defined by the presented specification of RTFD* has no state in which the determiner is obliged to declare the proponent as the winner while the proponent does not accept the topic of the argumentation.

PROOF. Assume that $s \supseteq \{Obliged(Det, Declare(Pro)), \neg Accepts(Pro, Topic)\}$ is a state of the transition system defined by D^{RTFD^*}. Since s is a state of the transition system defined by D^{RTFD^*}, s is an interpretation of σ^{rf} that satisfies $G \rightarrow F$ for every static law of the form '**caused** F **if** G' in D^{RTFD^*}. Given the static law (17), the only laws that make true the $Obliged(Det, Declare(Pro))$ fluent constant are static laws (15) and (19). Both of these laws require, amongst other conditions, that the $Accepts(Pro, Topic)$ fluent constant is true. However, according to our initial assumption $s \supseteq \{\neg Accepts(Pro, Topic)\}$. Therefore, s is not a state of the transition system defined by D^{RTFD^*}. □

5.5 Execution of RTFD*

In order to perform computational experiments with CCALC, we have to choose a concrete version of RTFD* with specific values for parameters such as deadlines, number of turns for each participant, and so on. We chose the following. (Other choices could of course have been made, and the experiments repeated for those.) First, we suppose that at most two actions are physically possible between two consecutive timeout events. Assume, for example, that only two messages can be exchanged in the given communication channel in the interval defined by any two consecutive timeout events. Second, to impose a limit on the length of each argumentation, we suppose that each agent has three turns to 'speak'—the disputation commences with the proponent's turn and finishes with the opponent's turn. Then it is the determiner's turn to declare the winner. Third, we assume that exactly one action takes place at each state transition (for example, it is not possible to perform concurrent actions). These extra constraints on the argumentation protocol allow us to determine the maximum number of transitions in any complete path of the transition system defined by D^{RTFD^*}. This is necessary for the submission of planning queries to CCALC. With the assumptions identified above the maximum number of transitions in a complete 'run' of this version of the argumentation protocol is twenty-one.

To test our formalisation of RTFD* we performed a number of queries with CCALC. The $C+$ specification of the chosen version of the argumentation protocol is translated into the input language of CCALC[2]. The computational experiments were based on the scenario used by Brewka [3,

[2]See [6] for details on the syntax of the input language of CCALC. For consistency reasons, we express the results of the queries in the notation that was employed (in this paper) to represent the social constraints of RTFD*.

pp.274–275] to illustrate his reconstruction of RTFD. This scenario concerns a legal dispute taken from US law: the proponent claims that its security interest in a particular ship is 'perfected' and the opponent denies this. Brewka's scenario includes argumentation about the priority of (the employed) default rules. Since our formalisation does not accommodate the prioritisation of default rules, this feature of the example was taken out. The formalisation that we tested includes five propositions and a default rule, and the topic of the argumentation is the *Perfected* proposition.

Query 1. We are in a state where the determiner is obliged to declare the proponent as the winner of the argumentation. In this state, may the determiner declare the opponent as the winner of the argumentation?

CCALC determines that the $Det{:}Declare(Opp)$ action is executable. Moreover, the resulting state holds the following information:

$$Winner(Opp);\ Obliged(Det, Declare(Pro));$$
$$Pow(Pro, Object(Det{:}Declare(Opp))$$

In the resulting state the winner of the argumentation is the opponent (see the *Winner* fluent constant). The obligation to declare the proponent as the winner of the disputation is due to either constraint (15) or constraint (19). This obligation is not discharged at the resulting state of the solution of this query because the determiner's declaration was invalid. Recall that a condition of both constraints (15) and (19) is the negation of the *FinalState* fluent constant. This constant becomes true when either the determiner performs a valid declaration or the final timeout takes place. If the determiner performed a valid declaration, then it would have made the *FinalState* constant true and, therefore, it would have discharged its obligation (the value of the remaining conditions of laws (15) and (19) is the same in the initial and resulting states of the query). In the resulting state the proponent is empowered to object to the invalid declaration—if it does so, the winner will no longer be the opponent.

Query 2. Given the initial state of the argumentation protocol, find the shortest path (if any) that ends in a state where some agent is empowered to object to a retracted proposition (or default).

CCALC finds the following solution (sequence of transitions):

$$Pro{:}Claim(Perfected);\ Timeout;\ Pro{:}Retract(Perfected)$$

The proponent's retraction is invalid because after the first timeout only the opponent is empowered to act. In other words, the value of $Pow(Pro, Retract(Perfected))$ is false at the penultimate state of the solution of the query because $Turn{=}\ Opponent$ (see constraint (7)). Notice that, although invalid, the proponent's retraction terminates its premise about *Perfected* that was earlier created by the action $Pro{:}Claim(Perfected)$. The premise was terminated because of constraint (11).

Query 3. Given the initial state of the argumentation protocol, is it possible to reach a state, within twenty-one transitions, where the determiner is obliged to declare the proponent as the winner while the proponent does not accept the topic of the argumentation?

CCALC finds no solution within twenty-one transitions. Since there is no solution within the maximum number of transitions, the determiner will never be obliged to declare the proponent as the winner while it (the proponent) does not accept the topic of the argumentation. The result of this query is consistent with Proposition 1.

6. DISCUSSION

Our specification of open agent systems was influenced by the work on *artificial social systems* [12]. The definition of *normative systems* [12, Definition 4.2] is similar to our definition of open agent systems. The definition of permitted actions in the artificial social systems approach requires that: (i) every agent knows what actions it is allowed to perform, and (ii) every allowed action is physically possible for each agent. Our definition of institutional powers, permissions and obligations does not require any of the two aforementioned properties.

The work on *commitment protocols* in MAS [18] bears many similarities to our work. Yolum and Singh formalise a set of operations on 'commitments' in the Event Calculus, employing an Event Calculus planner to facilitate the planning of agents that execute commitment protocols. We have also used versions of the Event Calculus for similar purposes (see, for example, [1]) though in this paper we have chosen to present our use of the $C+$ formalism.

Normative rules in *e-institutions* [5], like our specification of social constraints, provide an account of the obligations and sanctions of the members of open agent systems. Unlike social constraints, however, normative rules in [5] deal only with the communicative actions of the agents, and not with their physical actions.

Generally, work on the specification of open agent systems does not explicitly represent the institutional powers of the member agents. This point is the key difference of our work from related work in the literature: our specification of the social constraints explicitly represents the institutional powers of the agents, differentiates between institutional power, permission, physical capability and sanction, and employs formalisms with clear routes to implementation to provide a declarative representation of these concepts.

In principle, the kind of specifications presented in this paper could also be expressed in other temporal reasoning formalisms. The $C+$ language, however, has a number of important features that have led us to choose it as the basis for further developments. First, it is a comparatively expressive formalism with fine control (using '**inertial**' and other declarations) for specifying default persistence of fluent constants. The availability of static laws moreover means, amongst other things, that complex specifications can be given structure. For example, most of the rules defining $Pow(\dots)$ constants in this paper can be expressed as static laws with simple conjunctions as their conditions. In larger examples, as the conditions of such rules become increasingly complex, additional structure can be provided by introducing suitably chosen intermediate concepts, themselves defined by means of static laws. This is important if large specifications are to be undertaken. Second, besides its semantics through translation to a nonmonotonic 'causal theory' [6], a $C+$ action description has an explicit semantics in terms of transition systems. This is important because it provides a link to a wide range of other formalisms based on transition systems. This link is being exploited, for example, in an extended form of $C+$ that we have under development: the $(C+)^{++}$ language [16] includes direct support for (a version of) the 'counts as' relation for actions [11] and a treatment of per-

mitted/forbidden states, transitions and paths. We have not employed the $(C+)^{++}$ language in this paper partly because space prevents the presentation of its additional features and partly because the RTFD* protocol is too simple to make much use of them.

The language $C+$ (and its derivative $(C+)^{++}$) also has some important limitations. Most obviously, from a representational point of view, the language inherits the limitations of transition systems, in particular that the executable actions (transitions) in any given state s of the system, and their effects, can depend only on the state s and not on the path or history by which state s was reached. (Unless of course we encode the entire history in every state s—which can always be done but which does not produce a transition system representation as properly understood.) As noted in the text, in the case of the RTFD* protocol we were able to overcome this limitation by recording a small fragment of the history using the *ActionHappened* fluent constants to record past events. In other protocols where the moves available to a participant might depend on the entire history of the protocol so far (a participant can object to any of the actions of his opponent, for example, not just the immediately preceding one), the limitations of the transition systems model would be much more awkward to overcome. From the point of view of implementation, the CCALC environment provides an immediate and convenient means of implementing $C+$ action descriptions. The execution of the RTFD* specification, however, confirmed our previous experience regarding CCALC's efficiency, and in particular that it does not provide a practical means for supporting run-time ('on-line') activities[3]. It is possible to identify several ways to optimise its computation (see [2]), and of course CCALC is not the only means by which $C+$ action descriptions could be executed. The CCALC environment moreover does not provide any tracing or explanatory facilities. Clearly a natural explanation of why, for example, a given action was or was not valid in some given circumstance would be an invaluable feature for the kinds of applications we have in mind. These are directions for future research.

Finally, we outline two additional directions for further work. First, we need to extend our specification in order to cater for more complex normative relations such as duties and rights. Second, we are seeking to develop a more refined classification of 'valid' actions using a standardised vocabulary of terms such as 'proper', 'in order' and 'void'. Some suggestions along these lines can be found in [13].

7. ACKNOWLEDGEMENTS

This work has been undertaken in the context of the EU-funded ALFEBIITE Project (IST-1999-10298). We would also like to thank Joohyung Lee for his suggestions regarding the $C+$ language and CCALC.

8. REFERENCES

[1] A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In C. Castelfranchi and L. Johnson, editors, *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1053–1062. ACM, 2002.

[2] A. Artikis, M. Sergot, and J. Pitt. Specifying electronic societies with the Causal Calculator. In *Proceedings of Workshop on Agent-Oriented Software Engineering III (AOSE)*, LNCS 2585. Springer, 2003.

[3] G. Brewka. Dynamic argument systems: A formal model of argumentation processes based on situation calculus. *Journal of Logic and Computation*, 11(2):257–282, 2001.

[4] G. Brewka and T. Eiter. Prioritizing default logic. In *Festschrift 60th Anniversary of W. Bibel*. Kluwer, 1998.

[5] M. Esteva, J. Rodriguez-Aguilar, C. Sierra, P. Garcia, and J. Arcos. On the formal specifications of electronic institutions. In *Agent Mediated Electronic Commerce*, LNAI 1991. Springer, 2001.

[6] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, to appear in 2003. `http://www.cs.utexas.edu/users/vl/papers/nmct.ps`.

[7] E. Giunchiglia, J. Lee, V. Lifschitz, and H. Turner. Causal laws and multi-valued fluents. In *Proceedings of the Workshop on Nonmonotonic Reasoning, Action, and Change*, 2001.

[8] E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: Preliminary report. In *Proceedings of AAAI Conference*, pages 623–630. AAAI Press/The MIT Press, 1998.

[9] C. Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47:79–106, 1991.

[10] A. Jones and M. Sergot. On the characterisation of law and computer systems: The normative systems perspective. In *Deontic Logic in Computer Science: Normative System Specification*, pages 275–307. J. Wiley and Sons, 1993.

[11] A. Jones and M. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, 1996.

[12] Y. Moses and M. Tennenholtz. Artificial social systems. *Computers and Artificial Intelligence*, 14(6):533–562, 1995.

[13] H. Prakken. Formalising Robert's rules of order. Technical Report 12, GMD – German National Research Center for Information Technology, 1998.

[14] R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. The MIT Press, 2001.

[15] N. Rescher. *Dialectics: A Controversy-Oriented Approach to the Theory of Knowledge*. State University of New York Press, 1977.

[16] M. Sergot. The language $(C/C+)^{++}$. In J. Pitt, editor, *Deliverable D6(2) of ALFEBIITE EU-Project (IST-1999-10298)*, pages 55–84, 2002.

[17] R. Smith and R. Davis. Distributed problem solving: The contract-net approach. In *Proceedings of Conference of Canadian Society for Computational Studies of Intelligence*, pages 217–236, 1978.

[18] P. Yolum and M. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In C. Castelfranchi and L. Johnson, editors, *Proceedings of AAMAS*. ACM, 2002.

[3] Queries were computed in 40 seconds on a Pentium IV 2GHz, 1GB RAM computer running CCALC v.2.0b8.3.