Animated Specifications of Computational Societies

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

ABSTRACT

E-markets and negotiation protocols are two types of application domains that can be viewed as open computational societies. Key characteristics of such societies are agent heterogeneity, conflicting individual goals and limited trust. The risk that members of such societies will not conform to specifications imposes the need for a framework that will facilitate the designers to determine to what extent it is desirable to deploy their agents in such societies. We address this need by presenting a formal framework for specifying, animating, and ultimately reasoning about and verifying the properties of open computational systems. We view computational systems from an external perspective, aiming to account for the institutional and social aspects of these systems. We identify the key concepts and illustrate how they are used by formalising an example employing the contract net protocol. The framework and associated logical inferences have been implemented as a software platform that provides automated animation of the global states of an open system (society) during its execution. Simulations have demonstrated that the implementation of the framework establishes a foundation for a rich, formal representation of open computational societies.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Design, Experimentation

Keywords

Formal Specification, Norm-Governed Multi-Agent Systems

1. INTRODUCTION

E-markets and digital media rights management are two examples of application domains where software agents form computational societies in order to achieve their goals. Key characteristics of such societies are agent heterogeneity, conflicting individual goals, limited trust and a high probability of non-conformance to specifications. Consequently, it is of eminent importance that the activity of such societies is governed by a framework with formal, verifiable and meaningful semantics [16]. We address such a requirement by presenting a formal framework for specifying, animating, and ultimately reasoning about and verifying the properties of *open* computational societies/systems, i.e. systems where 'the behaviour of the members and their interactions cannot be predicted in advance' [5].

Several approaches have been proposed in order to provide formal tools for the specification and modelling of open agent systems. First, work that stems mainly from *philosophy* and the study of *legal systems* uses metaphors from human systems and behaviour in order to model computational systems and behaviour respectively. To our knowledge, there exists no implementation of these concepts in a computational level (in the context of agent societies). Second, research originating from the *distributed systems* field considers agents as computer processes and ascribes mental attitudes to them from an external perspective. Most of the frameworks in this category do not account for the legal or social concepts of agent systems. Third, work originating from *computational organisational theory* often views multi-agent systems (MAS) as computational organisations. Concepts like organisational rules and structures and institutionalised patterns of interactions are studied. However, 'too little has been done in organisation theory to supply precise qualitative models of norm-governed interaction' [12].

We describe a framework for the specification of computational systems that draws resources from these categories. We take a bird's eye view of the computational systems (as opposed to an agent's own perspective whereby it reasons about how it should act). We identify a number of key concepts of computational systems, namely the *social constraints*, *social roles* and *social states*. We specify these using a temporal formalism from AI (the *Event Calculus*, in the version presented here), and concepts from the formal study of legal and social systems (*institutionalised power*, *normative positions*), and distributed computing (*state transition systems*). In particular, we define three levels of specification for the social constraints:

- What kind of actions count as *valid actions*. Distinguishing between *valid* ('effective') and *invalid* actions enables the separation of meaningful from meaningless activities.
- What kind of actions are *permitted*. Determining the *per-*

AAMAS'02, July 15-19, 2002, Bologna, Italy

mitted, *prohibited*, *obligatory* actions enables the classification of the agent behaviour as 'legal' or 'illegal', 'ethical' or 'unethical', 'social' or 'anti-social', etc.
- What are the *sanctions* and *enforcement policies* that deal with 'illegal', 'unethical', 'anti-social' behaviour.

This kind of specification is the basis of a software platform that provides automated animation of the global states of agent systems. The information produced from the automated animation is available both to the society designer and the society members. This paper is divided in four main parts. The first presents a theoretical framework for the specification of open systems and the background work on which this framework is based. The second describes a specification and a logical animation of the execution of a contract-net protocol (by an open society). The third presents a computational framework that implements the theoretical one; the fourth discusses related and future work.

2. BACKGROUND

We differentiate between valid ('effective') and invalid actions with the use of a theory of *institutionalised power*. We follow Jones and Sergot [6] and view agent societies as instances of *normative systems* in the sense that agent interactions are described in terms of what these agents are permitted or obliged to do and in terms of other more complex normative relations between them. Our specification of permitted actions is motivated by a theory of *normative positions*. Finally, we reason about the agents' actions with the use of the *Event Calculus*.

2.1 Institutionalised Power

The term *institutionalised power*[1] refers to the standard feature of all norm-governed organisations whereby designated agents are empowered, by the institution, to create facts that have a *conventional significance* within that institution. Searle [13] for example has distinguished between *brute facts* and *institutional facts*. Being in physical possession of an object is an example of a brute fact (it can be observed); being the owner of that object is an institutional fact.

Jones and Sergot [7] present a formalisation of this concept in terms of an even more fundamental notion, viz., that within a given institution, certain kinds of acts or states of affairs have conventional significance, in that they *count as* other kinds of acts or states of affairs. For example, in an auction house, the auctioneer's performing the speech act 'the item x is sold' counts as, in the auction house, a way of establishing that item x is sold. One then says that the auctioneer has the power, or is empowered, within the auction house, to establish that item x is sold. The same action performed by an agent without this power has no effect. The owner of item x thereby acquires further powers: e.g. to transfer ownership of the item to another agent.

2.2 Normative Positions

The term *normative positions* refers to the range of possible normative relations that may exist between pairs (or larger collections) of agents with respect to performing a certain kind of action in any given circumstance. Examples of such relations are *right* (of one agent versus another), *duty* (of one agent to another), and so on. Sergot [14] presents a formal account of such relationships with references to the early work on social and legal theory on which it builds. In general, these are the kinds of normative relationships we want to be able to represent. For the purposes of this paper, we restrict attention to the simplest types of normative positions only: *obligations* and *permissions*. In [14] obligation is represented with the deontic operator O which expresses the idea that 'it is obligatory that'. Permission (represented by the deontic operator P) can be taken to be the dual of obligation, i.e. $PA =_{def} \neg O \neg A$. Further discussion of alternative treatments is beyond the scope of this paper.

2.3 Event Calculus

The *Event Calculus* (EC), introduced by Kowalski and Sergot [8], is a formalism that enables reasoning about events. It is based on a many-sorted first-order predicate calculus. *Events* (*actions*) in EC initiate and terminate *fluents*, which are properties that have different values at different points in time. The value of fluents is affected by the occurrence of events. A fluent starts to hold after the occurrence of an event that can initiate it. Similarly, a fluent ceases to hold after the occurrence of an event that can terminate it. The Event Calculus used in this paper is a subset of the 'full version' presented in [15].

3. SPECIFICATION OF OPEN SYSTEMS

We aim to provide a formal framework for executable specifications of open computational societies. We regard a computational society as *open* if the following properties hold:

1. The behaviour of the members and their interactions cannot be predicted in advance [5].
2. The internal architecture of the agents is not publicly known.
3. The members of the society do not necessarily have common goals, desires or intentions.

The first property implies that the execution of open societies is non-deterministic. Open societies are always subject to unanticipated outcomes in their interactions [5]. According to this property, the two remaining properties can be attributed to open agent societies. The second property implies that an open society can have members with different internal architectures. Therefore, we will treat open societies as *heterogeneous* ones. Moreover, we have no direct access to an agent's mental state and so we can only infer things about it. The third property implies that the members of an open society may fail to, or choose not to, conform to the (society) specifications in order to achieve their individual goals.

Often in the literature open agent societies are those where agents can enter or leave the society at any time [5]. We do not specify the way in which agents enter or leave societies. Usually, agents enter a society after having successfully executed a *role-assignment protocol*. The specification of this procedure/protocol is application-specific. Our definition of an agent society as open is irrespective of the protocols that specify the ways by which agents enter or leave societies.

We identify a number of concepts that characterise an open agent society of this type, now further structured as follows. What constitutes a society is a set of *agents*, a set of *constraints* on the society (norms, and other constraints, such as physical and logical constraints), a set of *roles* that members can play, the *state* of the members and the environment in which they act, a *communication language*, *re-*

[1]We will interchangeably use the terms 'institutionalised power' and 'power'.

Table 1: Fluents of the specification of open societies

Fluent	Meaning
$Valid(ag, act)$	ag performs valid action act
$Pow(ag, act)$	ag has the power to perform act
$Permitted(ag, act)$	ag is permitted to perform act
$Obliged(ag, act)$	ag is obliged to perform act
$Role\_of(r, ag)$	ag occupies role r
$Preconditions(r, ag)$	ag satisfies the preconditions of r
$Assign(r, ag)$	agent ag is assigned to role r

lationships between the members, including *ownership* and *representation* relations, and the *structure* of the society. In this paper we focus on the specification of the social states, social constraints and social roles. Due to the similarity of the term open societies to that of *institutions* [1, 12] (the only difference being that open societies are not necessarily goal-directed), we will use the terms institutional constraints and social constraints interchangeably (similarly we will interchangeably use institutional roles and social roles, and institutional states and social states).

3.1 Social Constraints

The semantics of an action is given by specifying in the Event Calculus what states of affairs that action can *initiate* and *terminate*. For example, the auctioneer's announcement of an item for sale initiates, among other things, the fact that the auction has started (provided of course that a set of further properties hold). Furthermore, this announcement terminates the power of the auctioneer to start an auction for the same goods (again provided that a set of further properties are satisfied).

Social constraints are also expressed by means of the Event Calculus. They are represented as a conjunction of *Initiates* and *Terminates* formulas (i.e. the domain description [15]) and as a conjunction of *HoldsAt* formulas (i.e. the state constraints [15]). With the use of the *Initiates* and the *Terminates* formulas the social constraints define the semantics of the agents' actions. Therefore, in different societies, or in different institutions within the same society, the same action might have different semantics (initiate/terminate different states of affairs).

We follow Jones and Sergot [7] and maintain the standard, in the study of social and legal systems, long established distinction between *permission*, *physical capability* and *institutionalised power*. For example, a bidder in a certain auction may be empowered to make a bid for an item, in the sense that if he does communicate the bid in the prescribed manner, it will count as a valid bid and must be taken into consideration by the auctioneer. But it may be that the bidder, though having the power to bid, is nevertheless not permitted to bid, in the sense that if he does bid he violates some norm that may be (temporarily perhaps) in place. And it could be that a bidder, though empowered and permitted to bid, may nevertheless be unable to bid, because he is (temporarily perhaps) unable to perform the physical act by means of which the bid is communicated.

Accordingly, we have three levels of specification of the *social constraints* of open systems. First, we specify what constitutes an action as being *valid*[2] ('well-formed', 'effective'). Second, we specify what actions (valid, invalid) are *permitted*. Based on permissions, prohibitions and obligations we can classify the behaviour of the agents into various categories such as 'social', 'anti-social' [9] etc. Third, we specify what are the *sanctions* that deal with anti-social behaviour and any other mechanisms designed to reduce it.

3.1.1 Valid Actions

On the first level of specification, we say that an action counts as a *valid* ('well-formed', 'effective') action at a point in time if and only if the agent that performed that action had the *institutionalised power* to perform it at that point in time. This constraint is formulated in EC as follows[3]:

$$\begin{aligned} HoldsAt(Valid(agent, action), time) \leftrightarrow \\ HoldsAt(Pow(agent, action), time) \wedge \\ Happens(Action(agent, action), time) \end{aligned} \quad (1)$$

Differentiating between valid and invalid actions is of great importance in the analysis of agent systems. For example, in an auction, the auctioneer has to determine which bids are valid and therefore, which bids are eligible for winning the auction.

3.1.2 Permitted Actions

The second level of specification defines which actions (valid, invalid or future ones) are *permitted*, *prohibited* or *obligatory*. This definition is *application-specific*. It *might* be that in a system an action is permitted at time t if and only if the agent that performed it had the power to do so at t:

$$\begin{aligned} HoldsAt(Permitted(agent, action), time) \leftrightarrow \\ HoldsAt(Pow(agent, action), time) \end{aligned} \quad (2)$$

In that case, invalid actions are not permitted and their performance may be considered 'anti-social'. In another setting, we might want to forbid an agent to perform an action even if he has the power to perform that action. For example, in an auction house the bidders have the power to bid when the auctioneer announces an item. However, we might want to forbid agents from bidding if they do not have the resources to pay for the bid. In other words, even if they are empowered to bid they might not be permitted to do so. The details are application-dependent. Similarly, the specification of obligations is application-specific. Determining what actions are permitted, prohibited or obligatory enables the classification of the behaviour of the agents and the system as social/anti-social, acceptable/unacceptable [9] etc.

3.1.3 Enforcement Policies

Finally, on the third level of specification, the definition of the *sanctions* and the *enforcement policies* is also application-specific. For example, in a contract-net protocol, the manager's failure to pay the agreed price to the awarded contractor by a particular time may create a sanction to the effect that the manager is no longer empowered to initiate a new contract-net protocol. In a different setting, the same type of misbehaviour might create a different sanction expressed in terms of some social concept such as bad reputation. Sanctions are one means by which an agent system can discourage unacceptable or anti-social behaviour.

[2]We use the term *valid* in this paper acknowledging the fact that this term may not be appropriate in all domains.

[3]Variables (denoted by lowercase first letter) are assumed to be universally quantified unless otherwise indicated.

Another mechanism is to try to devise additional controls (physical or institutional) that will force agents to comply with their obligations (for example, certain bids are physically blocked).

Institutional powers, normative positions and sanctions are ascribed to the agents from an external perspective. Agents are usually aware of their institutional powers (otherwise they cannot make plans for creating institutional facts, such as *ownership relations*) but they are not necessarily aware of the permissions associated with them. Furthermore, it might be the case that agents have to comply with a different set of *internal* norms, imposed on them by their designers (say). These are part of the internal architecture of the agents. In this paper, we deal only with *external* normative positions and sanctions, i.e. those ascribed to the agents from an external perspective.

3.2 Social Roles

A social role r is defined as the set of *preconditions* that an agent must satisfy in order to occupy r, and the set of *constraints* that describe the behaviour of the agent while occupying r.

The preconditions of a role are properties that an agent must have in order to occupy that role. Such properties may include a set of competences. We illustrate the concept of role preconditions by describing a *role-assignment protocol* (as mentioned earlier, agents usually execute a role-assignment protocol before entering a society). A simple role-assignment protocol can be briefly described as follows[4]: A role-assigning authority (e.g. a trusted third party) announces a role to a set of agents and specifies the preconditions that the agents should satisfy in order to be eligible to get the role. The agents that are interested in occupying that role apply for it specifying whether or not they satisfy the role preconditions. Finally, the role-assigning authority decides which agents will be assigned the role and which agents will not. The decision of the role-assigning authority will be mainly based on two factors: (i) whether an agent satisfies the role preconditions or not, and (ii) whether the *role-assignment constraints* are violated or not. The role-assignment constraints constrain the assignment of roles and are defined in an application-specific manner. For example, in an auction protocol, the role-assignment constraints would specify that only one agent should occupy the role of the auctioneer. Therefore, the power of the role-assigning authority during the execution of a role-assignment protocol (that takes place before the commencement of an auction) is formalised as:

$$\begin{aligned} &HoldsAt(Pow(authority, Assign(Auctioneer, agA)), time) \leftarrow \\ &\quad HoldsAt(Role\_of(TTP, authority), time) \wedge \\ &\quad HoldsAt(Preconditions(Auctioneer, agA), time) \wedge \\ &\quad \neg HoldsAt(Role\_of(Auctioneer, agB), time) \end{aligned} \tag{3}$$

Constraint (3) states that a role-assigning authority (e.g. a trusted third party (TTP)) has the power to assign the role of the auctioneer to an agent (say agA) if (i) agA satisfies the preconditions of the role (see Table 1) of the auctioneer (we assume that agA has applied for that role), and (ii) no other agent has already been assigned the role of the auctioneer. As far as the role preconditions are concerned, it is sometimes desirable to impose the following constraint:

$$\begin{aligned} &HoldsAt(Preconditions(rolename, agent), time) \leftarrow \\ &\quad HoldsAt(Role\_of(rolename, agent)), time) \end{aligned} \tag{4}$$

(4) states that it is a constraint of the system that an agent satisfies the preconditions of a role as long as he occupies that role. This simplifies the construction of some computational societies, though it must be noted that it does not apply in general.

The constraints that describe the behaviour of an agent occupying role r specify the powers and normative positions that are associated with r. These constraints are expressed as a conjunction of *Initiates*, *Terminates* and *HoldsAt* formulas. The first two types of formulas state what states of affairs an agent occupying role r can initiate and terminate due to the fact that he occupies r. The third type of formulas states what are the powers and permissions that an agent occupying the role r has due to the fact that he occupies r. The set of social constraints is a super-set of each set of role constraints. In particular, the set of social constraints is the union of the sets of constraints characterising each role combined with the set of role-assignment constraints and the set of *role-independent* constraints i.e. constraints that apply to all members of a society, irrespective of the roles they occupy.

3.3 Social States

Time is taken to be finite, discrete, linear, with time points ranging over the natural numbers. Temporal intervals are defined as pairs of points. The next time point of t is $t+1$. The global state of a multi-agent system is represented in a similar way to Fagin et al. [3]. The set of global states of a multi-agent system is defined as $S \subseteq L_e \times L_1 \times ... \times L_n \times L_{in_1} \times ... \times L_{in_k}$. A global state $s = (l_e, l_1, ..., l_n, l_{in_1}, ..., l_{in_k})$ represents a snapshot of the multi-agent system in which agent m is in state $l_m \in L_m$ the institution in_j is in local state $l_{in_j} \in L_{in_j}$, and the environment in state $l_e \in L_e$. Within each society there are other groupings which (following standard usage [1, 12]) we call 'institutions'. Such institutions have their own constraints, roles, communication language etc. We model the local states of the institutions that are part of the greater society. The local states of the institutions contain information like the institutional powers and the normative positions of their agents. Therefore, the focus is on the local states of the institutions (that are part of the society) rather than the local states of the agents. However, the example treated in this paper contains (or can be viewed as containing) only one institution. Consequently, for the purposes of this paper we will only deal with the agents' local states. The term global state is interchangeably used with the term global social state.

4. SPECIFICATION OF A CNP

In order to illustrate the way we specify and animate agent societies, we describe, specify and animate in this section a society that executes a variation of the contract-net protocol (CNP) [17]. In [10] we describe an abstract producer/consumer (APC) scenario where producers sell information to consumers. The producers are explorer agents that map out the distribution of oil in their environment and consumers are cartographer agents that initiate contract-net

[4]This is just an example of a role-assignment protocol. The specification of such kinds of protocols is application-specific.

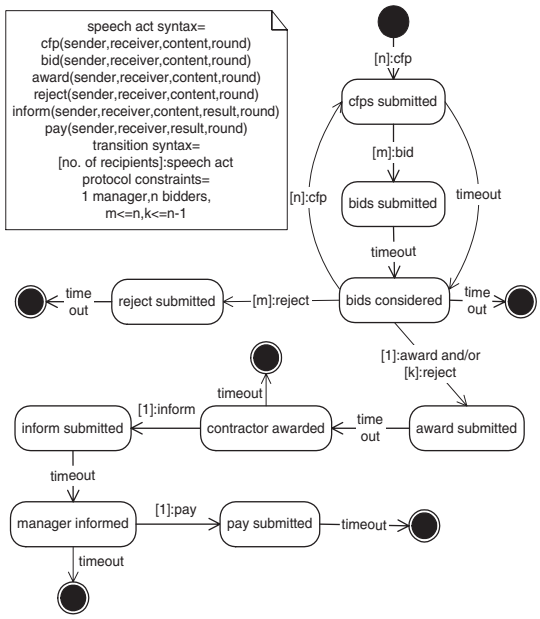

Figure 1: The State Transition Diagram of the CNP.

protocols to acquire the maps from the explorers. The main roles of the society that executes the contract-net protocol are that of the *Cartographer* and that of the *Explorer* (we will intermittently call cartographers managers and explorers bidders or potential contractors). A brief description of the CNP (Figure 1) is the following: a manager issues a call for proposals (*Cfp*) for a particular task to a set of bidders. Bidders submit their bids (if they are interested) to the manager. The manager then has three choices: (i) award a particular bid, (ii) reject the received bids, or (iii) issue a new *Cfp* incrementing the *protocol round*[5]. In the first two cases the protocol ends. In the third case the protocol starts again. Actions must be performed according to specified deadlines/timeouts.

It is important to outline a number of assumptions that we make in order to facilitate our analysis. Our main assumption is that when an agent sends a message to another agent, the recipient will always receive the message (i.e. no messages are lost). In addition, all messages are received at the same time-point that they were sent. Furthermore, in the analysis followed below we assume that a role-assignment procedure has already been completed. These assumptions are made in order to abstract from low-level details such as the reliability of the message-passing system and focus on our main objective which is the specification and the representation of the CNP.

4.1 Social Constraints

In this variation of the CNP we do not define any role-independent constraints. The set of all social constraints in this example is the union of the cartographer-related constraints and the explorer-related constraints. We describe the social constraints, and thereby implicitly describe all role constraints.

4.1.1 Valid Actions

First, we specify the powers of the agents that execute the CNP and therefore their ability to perform valid ('effective') actions. Figure 1 illustrates all possible valid actions[6] that the agents of the CNP can perform and the state transitions that these actions create. At the initial state (before the commencement of the CNP) the agent that occupies the role of the *Cartographer* (represented by the variable c) has the power to issue a call for proposals (*Cfp*) to all the agents that occupy the role of the *Explorer* (represented by the variable e). This is formulated as follows:

$$\begin{aligned} &InitiallyP(Pow(c, Cfp(c, e, content, 1))) \leftrightarrow \\ &\qquad HoldsAt(Role\_of(Cartographer, c), 0) \wedge \\ &\qquad HoldsAt(Role\_of(Explorer, e), 0) \end{aligned} \tag{5}$$

Issuing a valid *Cfp* to a set of bidders empowers these bidders to issue a valid *Bid*. In other words:

$$\begin{aligned} &Initiates(Cfp(c, e, content, round), \\ &\quad Pow(e, Bid(e, c, relatedContent, round)), time) \leftrightarrow \\ &\quad HoldsAt(Valid(Cfp(c, e, content, round)), time) \wedge \\ &\quad Matches(content, relatedContent) \end{aligned} \tag{6}$$

The *Matches* predicate specifies whether two task descriptions are related or not and is defined in an application-specific manner. A valid *Cfp* terminates, among other things, the power of the cartographer to issue another valid *Cfp*:

$$\begin{aligned} &Terminates(Cfp(c, e, content, round), \\ &\quad Pow(c, Cfp(c, e, content, round)), time) \leftrightarrow \\ &\quad HoldsAt(Valid(Cfp(c, e, content, round)), time) \end{aligned} \tag{7}$$

When the CNP commences (this happens when the cartographer issues a valid *Cfp*) a global clock starts ticking. The cartographer will have the power to *Award* or *Reject* any of the incoming valid bids only when the first timeout elapses. Therefore, a valid *Bid* does not immediately empower the cartographer to issue a valid *Award* or *Reject*. The timeout initiates that power. Timeouts are viewed as events (with semantics in the context of the states of affairs they initiate and terminate) even though they are not performed by a member of the society. All the powers are propagated in the manner outlined above. If an agent performs an invalid action then this action is recorded but it has no effect (in this example) on the state transition mechanism.

4.1.2 Permitted Actions

We now specify what actions are permitted, prohibited and obligatory during the execution of the CNP. For this variation of the CNP permissions are specified by (2), i.e. an agent is permitted to perform an action if and only if he has the power to perform that action. This would not hold in general, as explained earlier. Obligations arise in three situations. First, issuing a valid *Award* obliges the

[5]The protocol round is a parameter of the description of the actions that agents perform. It is represented by an integer that, initially, is equal to 1 and is incremented by 1 each time a cartographer performs a valid *Cfp*. Alternative formalisations may omit the protocol round.

[6]From this point onwards we will represent valid actions as a fluent with only one parameter, the action description. Constraint (1) is trivially modified to incorporate this change.

Table 2: Animation of a run of the CNP.

Time	Events	agent E_1	agent E_2	agent C
0		∅	∅	Pow(C,Cfp(C,{E_1,E_2},content, round))
1	Cfp(C,{E_1,E_2},Map(x),1)	Pow(E_1,Bid(E_1,C, relatedContent,1))	Pow(E_2,Bid(E_2,C, relatedContent, 1))	∅
2	Bid(E_1,C,Map(w),1), Bid(E_2,C,Map(z),1)	∅	∅	∅
3	Timeout	∅	∅	Pow(C, Award(C,E_1,Map(w),1)), Pow(C, Award(C,E_2,Map(z),1)), Pow(C, Reject(C,E_1,Map(w),1)), Pow(C, Reject(C,E_2,Map(z),1)), Pow(C,Cfp(C,{E_1,E_2},newContent,2))
4	Reject(C,E_2,Map(z),2)	∅	∅	Pow(C, Award(C,E_1,Map(w),1)), Pow(C, Reject(C,E_1,Map(w),1))
5	Award(C,E_1,Map(w),2)	∅	∅	∅
6	Timeout	Pow(E_1,Inform(E_1,C, Map(y),result,2))	∅	∅
7	Inform(E_1,C,Map(y), Map_of(y),2)	∅	∅	∅
8	Timeout	∅	∅	Pow(C, Pay(C, E_1, Map_of(y), 2))
9	Pay(C, E_1, Map_of(y), 2)	∅	∅	∅
10	Timeout	∅	∅	∅

cartographer to reject all remaining valid bids:

$$\begin{aligned} &Initiates(Award(c, e, content, round),\\ &\quad Obliged(c, Reject(c, e2, content2, round)), time) \leftrightarrow\\ &\quad HoldsAt(Valid(Award(c, e, content, round)), time) \wedge\\ &\quad HoldsAt(Valid(Bid(e2, c, content2, round)), time2) \wedge\\ &\quad \neg(e = e2) \end{aligned} \quad (8)$$

Second, at the *contractor awarded state* (see Figure 1), the awarded contractor is obliged to inform about the outcome of the awarded task. Third, at the *manager informed state* the cartographer is obliged to pay the awarded contractor. Here we have chosen to take the view that these constraints are to be regarded as norms. An alternative would be to view them instead as part of the specification of what counts as a valid action. In that case, any failure to comply with these requirements would invalidate the whole protocol (both interpretations are possible; we have picked one for the sake of the example).

4.1.3 Sanctions

In this specification of the CNP sanctions arise when agents do not comply with their obligations. For example, since we are viewing the obligation as a norm, the awarded contractor's failure to issue a valid *Inform* by the specified time, creates a sanction to the effect that the awarded contractor is no longer empowered to enter another CNP. Similarly, the end-state that is reached when a timeout occurs at the *manager informed* state includes a sanction that defines that the manager is no longer empowered to enter a new CNP.

4.2 Social States

Consider a CNP with two agents, a cartographer C and an explorer E. Moreover, the following fluents characterise C at a particular point in time in a run of the CNP:

- $l_C \supseteq \{Pow(C, (Cfp(C, E, content, round)))\}$.

At this time point C issues a *Cfp* (no other action takes place). The effects of this action can be described as follows:

- $l'_C \supseteq \{\neg Pow(C, (Cfp(C, E, content, round)))\}$. This is due to constraints (1) and (7).
- $l'_E \supseteq \{Pow(E, Bid(E, C, relatedContent, round))\}$. This is due to constraints (1) and (6).

In this scenario we choose to represent in the state of the environment the *conversation state* [16]. In this specification of the CNP we represent the protocol round, the content (of the awarded task, if any), the contracts (a valid *Award* creates a contract) and the result of the awarded task in state of the environment.

4.3 Animation of the CNP

Given a set of events and a temporal ordering of them, we can find out with the use of the specification of the CNP the set of powers, permissions, obligations, sanctions of each agent at each point in time. We show here the animation of one run of the CNP by a group of 3 agents; two explorers E_1, E_2 and a cartographer C. A sketch of this run is the following: C performs a *Cfp* and then the two explorers submit their bids. Then the cartographer rejects the bid of E_2 and awards the bid of E_1. E_1 informs about the result of the awarded task and C pays him the specified price.

Table 2 illustrates the way the local states change over time (due to space limitations Table 2 only shows the powers of the agents). The state transitions are based on the events of each time point. For example, initially, at time $t = 0$, because of constraint (5), the cartographer is empowered to issue a *Cfp* and the explorers have no powers. At time $t = 1$ the cartographer performs a *Cfp* and the values of the fluents that characterise the participants change as described in section 4.2. That is, as a result of the cartographer's action, the explorers are empowered to issue valid bids (these new powers are shown for each of the explorers). Furthermore, at the same point the display for the cartographer does not include the fact that he is empowered to issue another *Cfp*. Since constraint (2) defines the permissions in this example, the agents have the same permissions as powers. The full display of the participants includes fluents that describe their obligations, sanctions and roles.

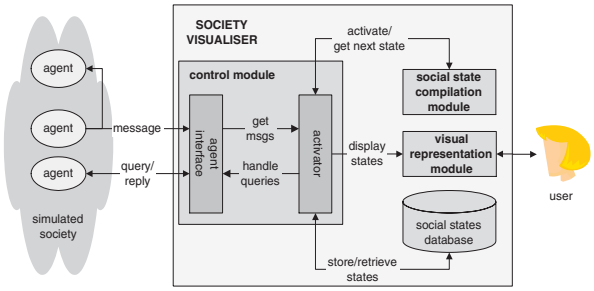

Figure 2: The Society Visualiser.

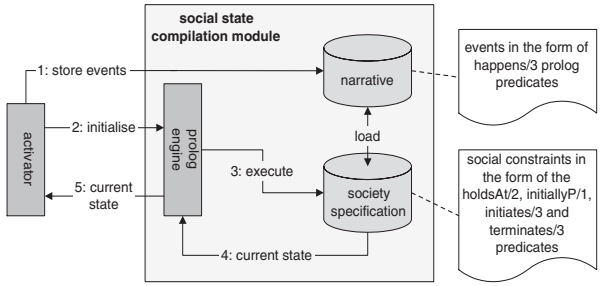

Figure 3: Social State Compilation.

5. THE SOCIETY VISUALISER

In this section we describe a computational framework called *Society Visualiser* (SV), that builds upon the theoretical framework for the specification of open systems and that compiles (produces) and updates the global state of an agent society. First we describe the architecture of the SV and then we demonstrate the way the SV produces automated animation of the runs of the CNP.

The basic architecture of the SV is illustrated at the right-hand side of Figure 2. The SV includes three main components: the *control* module, the *social state compilation* module and the *visual representation* (of the social states) module. A brief description of the functionality of the SV is the following: the control module mainly receives *report* messages from the members of the agent society. When a member of the society sends a message to a peer, he also sends that message to the control module of the SV for monitoring purposes. These messages are called *report* messages and are necessary for the compilation of the social states. Given the report messages (and all other monitored events), the control module activates the social state compilation module in order to produce the local states of the institutions and the environment at each point in time. It is worth noting that not all report messages will modify the global state, e.g. an event might not necessarily initiate/terminate a power, a normative position or a sanction. In any case, the local states of the institutions (here only one), as well as that of the environment are stored in a database and are displayed in a graphical interface for the benefit of the society designer. It is convenient to display social/institutional facts relating to each agent separately.

Agents can also send *query* messages to the SV in order to find out the social state of the group or of their peers. The control module of the SV handles such messages as soon as they arrive and an appropriate reply is sent to the originating agent.

5.1 Control Module

The control module contains two main sub-components, the *agent interface* component and the *activator* component (Figure 2). The agent interface receives the messages sent by the members of the society and either stores them attaching a timestamp (in the case of report messages) or forwards them to the activator (in the case of query messages). The control loop of the activator module operates as follows: the activator retrieves the current local state of each institution from the social state compilation module. The compilation of the social state is demonstrated in Figure 3 and described in the next section. The current local state is stored in a database and displayed in a graphical interface. Finally, the activator stops its execution for a specified period of time and then retrieves the new report messages that the agent interface module has received.

5.2 Social State Compilation Module

The main sub-component of the social compilation module is the *society specification database* (Figure 3). This database holds the social constraints based on which the society can change states. These constraints are represented in the form of the Event Calculus [15] in the `Prolog` programming language. In particular, the society specification database contains, among other things, the *domain description*, the *initial state* and the *domain/state constraints*. The domain description specifies the `initiates/3` and `terminates/3` predicates. The initial state specifies the `initiallyP/1` predicates describing what holds before the commencement of the execution of the society. The domain constraints specify the `holdsAt/2` predicates. Another component of the social state compilation module is the *narrative* module. This module describes what has happened in the society with `happens/3` predicates.

Figure 3 demonstrates what happens when the activator attempts to retrieve the current social state. The activator stores in the narrative the set of incoming messages of the last interval (along with other events like timeouts), and then initialises the `Prolog` engine. The `Prolog` engine loads (i) the predicates that represent the social constraints (that are stored on the society specification database) and (ii) the predicates that represent the externally observable events (that are stored in the narrative module). Then, it calls the following predicate (for each institution):

```
holdsAt(local_state(institution_name,State),time).
```

The result of that call is a set of fluents (represented by the `State` variable) that is returned to the `Prolog` engine and consequently to the activator. This set of fluents represents the local state of `institution_name` at `time`.

The society specification database is the only component of the SV that is application-specific. Therefore, when animating the global states of different societies only the society specification component will change and the remaining components of the SV will remain exactly the same.

5.3 Visual Representation Module

As soon as the control module retrieves the social state of a particular time-point, it displays it in a graphical interface for the benefit of the user/society designer. In this graphical display (Figure 4) the social state of each agent and institu-

tion is divided in five categories: (i) powers, (ii) permissions, (iii) obligations, (iv) sanctions and (v) valid actions. For example, when the user selects the powers of the members of institution `in1` at `time`, then the following query is executed in the social state compilation module:

```
holdsAt(pow(in1, Powers), time).
```

Furthermore, the local state of the environment is divided into a number of categories that each represent the variables of the conversation state.

5.4 Automated Animation of the CNP

The specification of the CNP has been implemented in the Society Visualiser. As mentioned earlier, the only component of the SV that is application-specific is the *society specification database*. Therefore, we will briefly describe the society specification database. This database consists of the social constraints of the CNP. For example, constraint (7) is represented in the society specification database as:

```
terminates(cfp(C,E,Content,Round),
  pow(C,cfp(C,E,Content,Round)),Time):-
holdsAt(valid(cfp(C,E,Content,Round)),Time).
```

As far as the domain constraints are concerned, the only constraint that is CNP-specific is the one that specifies the local state of the environment. We have animated a number of different runs of the execution of a CNP with the use of the SV. These included 4 agents; `cartographer1` occupying the role of the `cartographer` and `explorer1`, `explorer2` and `explorer3` occupying the role of the `explorer`. Figure 4 demonstrates the GUI of the SV at a particular point in time of a particular run of the CNP. The display of `cartographer1` includes the fact that he is empowered to issue a `cfp`, thus starting a new protocol round. In addition, he is empowered to either `award` or `reject` the bids of `explorer1` and `explorer2`. The display of this agent is similar to the display of the cartographer agent C at time point 3 as described in Table 2. In other words, the display of `cartographer1` includes these powers because the previous execution of the CNP (time points 0-6) is similar to the one described by Table 2. That is, `explorer1` and `explorer2` have performed valid bids and a `timeout` has taken place (see Figure 1). The display of `cartographer1` (see Figure 4) demonstrates that he is permitted to perform the same actions that he is empowered to perform. This is true because, as we mentioned in section *4.1.2*, permitted actions are specified in this CNP by constraint (2).

The SV has been tested with the use of different narratives describing different runs of the CNP. These narratives describe several types of agent (mis-)behaviour like issuing invalid acts, non-compliance to obligations etc. Furthermore, we have modified some of the assumptions that were presented in this paper. We have forbidden the performance of valid actions, modified the definition of sanctions etc.

6. RELATED WORK

We review related work from both the theoretical and the computational perspective of the specification and animation of open systems. Considering the theoretical perspective, the following frameworks are closely related to the aims of our work: Interpreted Systems [3], Artificial Social Systems [9], e-institutions [2], commitment protocols [16], the AALAADIN model [4] and the Gaia methodology [20]. The work on Interpreted Systems and Artificial Social Systems mainly focus on ascribing mental attitudes to the agents from an external perspective. We follow these approaches on the representation of the global state of MAS. The notion of commitment in [16] has some similarities to the notion of obligation in our framework. Esteva et al. [2] devise a formal specification language to design open agent systems as *electronic institutions*. In this approach, *roles* are defined as patterns of behaviour, *normative rules* are specified to limit or enlarge the space of agent actions and *scenes* are defined in order to represent the different contexts within an organisation in which agents can interact. Scenes can be viewed as sub-societies in [9]. The society executing the CNP that was described in this paper can be viewed as one scene. Related scenes would be the role-assignment procedure and possibly auctions that could occur in parallel with the CNP. Esteva et al. provide an informal account of notions such as duties, obligations and commitments. In a similar manner, the Gaia methodology provides an informal account of concepts such as responsibilities, permissions and rights.

As far as the computational perspective is concerned, *FishMarket* [11] and *MyWorld* [19] bear some similarities to the SV. Fishmarket, although primarily concerned with trading strategies and market conditions, also addresses issues that are associated with the social structure of the simulated societies. However, there is a lack of formalisation of concepts such as rights, obligations and social relations as well as an explicit representation of them during simulations. MyWorld represents an operationalisation of the *Agent0* framework; to this end, it facilitates the modelling of several aspects of social agent behaviour (e.g. intentions and commitments), but it does so by considering the internal architecture of the agents.

7. DISCUSSION

A formal theory of multi-agent systems (MAS) and, therefore, of open systems should provide the following [18]:

- A tool that enables the specification and verification of MAS.
- A tool that enables reasoning about MAS.
- A basis upon which richer theories and concepts, like socio-cognitive phenomena, can be formally described and tested.

Our framework describes a formal specification of some of the key concepts of open systems with the use of well-established theories. A complete specification of open systems requires the specification of the remaining concepts, like social structure [2, 9], explicit representation of the ACL (e.g. as in [2, 4, 16, 20]), representation relationships etc. The animation of a run of the CNP showed that our framework provides a rich representation of the execution of agent systems and, consequently, can be used for reasoning about the properties of these systems. Finally, this representation of agent systems can be (and is currently) used for the analysis of rich theories (e.g. socio-cognitive phenomena). In particular, the SV is being currently used for a series of trust experiments in the context of the ALFEBIITE EU project. The aim of these experiments is to explore computational analogues of socio-cognitive theories including formalisations of trust and reputation. In these experiments the SV provides the experiment designer with a global representation of the system. Furthermore, the members of the

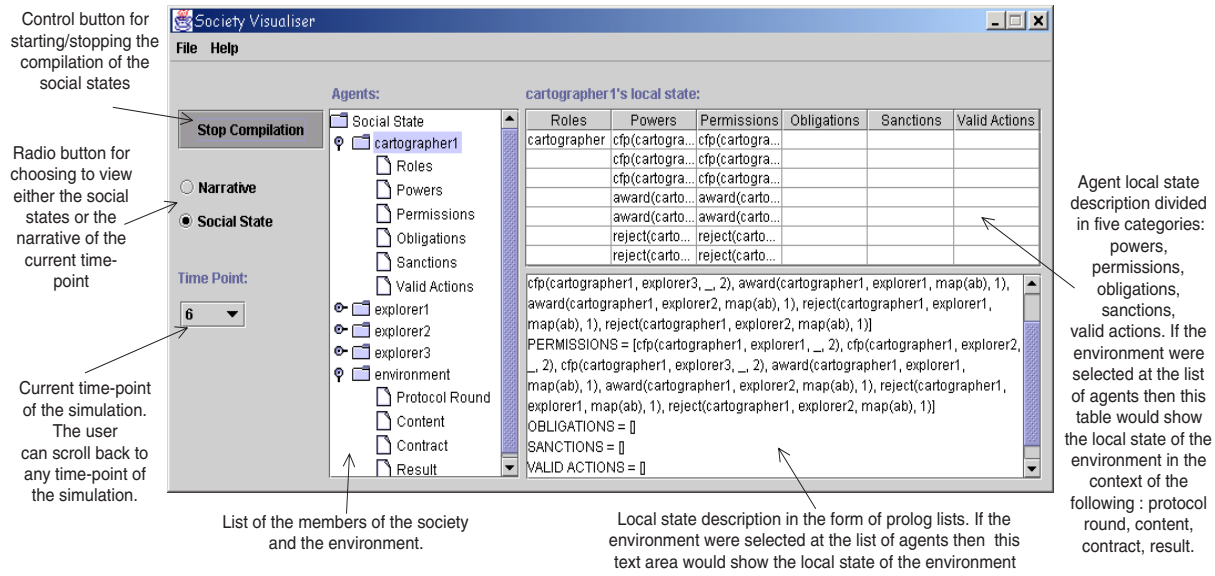

Figure 4: The main GUI of the SV.

society can query the SV about the social state of their peers and therefore, have an additional parameter in their trust update mechanism.

We aim to facilitate agent designers in deciding whether it is desirable or not to deploy their software agents in societies where the heterogeneous members may not conform to the specifications in order to achieve their individual and possibly competitive goals. To achieve that, first we need to have a formal specification of the computational systems that is based only on externally observable information. Second, based on the formal specification we need to able to reason about and ultimately verify properties (like safety, fairness, liveness) of the computational systems. In this paper we presented a way of providing a formal specification of agent systems. The next step is to investigate methods for the verification of the properties of these systems. We are currently working on a new version of this framework which replaces the use of the Event Calculus by a formalism with a more explicit transition system semantics.

8. ACKNOWLEDGEMENTS

This work has been undertaken in the context of EU-funded ALFEBIITE Project (IST-1999-10298). We would like to thank Lloyd Kamara for his help in the development of the Society Visualiser and for his invaluable comments during the writing of the paper. We have also benefitted from the contributions of Andrew Jones to the concept of roles.

9. REFERENCES

[1] K. Carley and L. Gasser. Computational organisational theory. In *Multi-Agent Systems: A Modern Approach to DAI*, page 300. MIT Press, 1999.

[2] M. Esteva, J. Rodriguez, C. Sierra, P. Garcia, and J. Arcos. On the formal specifications of electronic institutions. LNAI 1991, pages 126–147. 2001.

[3] R. Fagin, J. Halpern, Y. Moses, and M. Yardi. *Reasoning About Knowledge*. The MIT Press, 1995.

[4] J. Ferber and O. Gutknecht. Operational semantics of multi-agent organisations. In *Intelligent Agents IV*. 2000.

[5] C. Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47:76–106, 1991.

[6] 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*. Wiley, 1993.

[7] A. Jones and M. Sergot. A formal characterisation of institutionalised power. *IGPL*, 4(3), 1996.

[8] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–96, 1986.

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

[10] J. Pitt, L. Kamara, and A. Artikis. Interaction patterns and observable commitments in a multi-agent trading scenario. In *Autonomous Agents 2001*, 2001.

[11] J. Rodriguez-Aguilar, F. Martin, P. Noriega, P. Garcia, and C. Sierra. Towards a test-bed for trading agents in electronic auction markets. In *AI Communications*, pages 5–19, 1998.

[12] F. Santos, A. Jones, and J. Carmo. Action concepts for describing organised interaction. In *HICSS*, 1997.

[13] J. Searle. What is a speech act? In *Philosophy of Language*, pages 130–141. 1965.

[14] M. Sergot. A computational theory of normative positions. *ACM Transactions on Computational Logic*, 2(4):522–581, 2001.

[15] M. Shanahan. The event calculus explained. *Artificial Intelligence Today*, pages 409–430, 1999.

[16] M. Singh. A social semantics for agent communication languages. In *Issues in Agent Communication*, volume 1916 of *LNCS*, pages 31–45. 2000.

[17] R. Smith and R. Davis. Distributed problem solving: The contract-net approach. In *2nd Conference of Canadian Society for CSI*, pages 217–236, 1978.

[18] M. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, UMIST, 1992.

[19] M. Wooldridge. This is MyWorld: The logic of an agent-oriented DAI testbed. In *Intelligent Agents*. Springer-Verlag, 1995.

[20] M. Wooldridge, N. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *AAMAS Journal*, 3(3):285–312, 2000.