Specifying Electronic Societies with the Causal Calculator

Alexander Artikis[1], Marek Sergot[2] and Jeremy Pitt[1]

Imperial College of Science, Technology and Medicine, London, UK
[1]Electrical Engineering Department, SW7 2BT, +44 (0)20 75946221
[2]Department of Computing, SW7 2BZ, +44 (0)20 75948218
a.artikis@ic.ac.uk, mjs@doc.ic.ac.uk, j.pitt@ic.ac.uk

Abstract. In previous work [1] we presented a framework for the specification of open computational societies i.e. societies where the behaviour of the members and their interactions cannot be predicted in advance. We viewed computational systems from an external perspective, with a focus on the institutional and the social aspects of these systems. The social constraints and roles of the open societies were specified with the use of the Event Calculus. In this paper, we formalise our framework with the use of the $C+$ language, a formalism with explicit state transition semantics. We use the implementation of the $C+$ language, the `Causal Calculator`, a software tool for representing commonsense knowledge about action and change, to animate and validate the specifications of computational societies. We demonstrate the utility of the `Causal Calculator` (by specifying and executing a Contract-Net Protocol) and comment on its functionality regarding the specification of computational societies.

1 Introduction

Negotiation protocols [2, 11] and Virtual Enterprises [6] are two examples of application domains where software agents form computational societies in order to achieve their possibly competing goals. Key characteristics of such societies are agent heterogeneity, unpredictable behaviour [7], conflicting individual goals, limited trust and a high probability of non-conformance to specifications. Consequently, it is important that the activity of such societies is governed by a framework with formal and meaningful semantics [16]. In order to address such a requirement we presented in [1] a framework for specifying, animating, and ultimately reasoning about and verifying the properties of open electronic societies, i.e. computational systems where the behaviour of the (heterogeneous and possibly competing) members and their interactions cannot be predicted in advance.

The framework for the specification of e-societies was formalised with the use of the Event Calculus [14]. In this paper we formalise the framework with the use of the $C+$ language [5], a formalism with *explicit* state transition semantics.

We use the `Causal Calculator` (`CCALC`), a software tool that implements $C+$, to execute and, prove properties of, the specifications of open societies.

This paper is structured as follows. First, we briefly describe the $C+$ language and the `Causal Calculator`. Second, we present the framework for the specification of open e-societies. Third, we specify a Contract-Net Protocol (CNP) [17] with the use of the framework for the specification of e-societies and the $C+$ language. Fourth, we demonstrate the utility of `CCALC` by executing and 'validating' the specifications of the CNP. Finally, we discuss related work, summarise and comment on the functionality of `CCALC` regarding the specifications of computational societies.

2 The $C+$ language

The action language $C+$ [5] enables the representation of properties of actions, including actions with conditional and indirect effects and concurrently executed actions. An *action description* in $C+$ is a set of $C+$ rules that define a transition system of a particular kind. In this section we briefly present $C+$[1].

The representation of an action domain in $C+$ consists of *rigid* constants, *fluent* constants and *action* constants:

- *Rigid constants* are symbols that represent the features of the system whose value is fixed and does not depend on the state.
- *Fluent constants* are symbols characterising a state. They are divided in two categories: *Simple fluent constants* and *statically determined fluent constants*. Simple fluent constants are related to actions by *dynamic rules* (i.e. rules describing a transition from a state s_i to its successor state s_{i+1}). Statically determined fluents are characterised by *static rules* (i.e. rules describing an individual state) relating them to other fluents (static and dynamic rules are defined below).
- *Action constants* are symbols characterising state transitions. Intuitively, they represent the actions of the agents and the environment.

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 of the form

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

- Every constant occurring in F or G is rigid or fluent. In other words, F and G are formulas of signature σ^{rf}.
- If F contains a rigid constant then every constant occurring in F, G is rigid.

In a static law fluents F and G are evaluated on the same state of the transition system. A dynamic law is an expression of the form

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

[1] We follow the formal and more detailed analysis of $C+$ provided in [5].

- Every constant occurring in F is a simple fluent constant (i.e. F is a formula of signature σ^{rf}).
- Every constant occurring in G is rigid or fluent (i.e. G is a formula of signature σ^{rf}).
- H is any combination of rigid constants, fluent constants and action constants (i.e. H is a formula of signature $\sigma^{rf} \cup \sigma^{act}$).

In a transition from state s_i to state s_{i+1} simple fluent constants in F and rigid and fluent constants in G are evaluated on s_{i+1} and rigid, fluent and action constants in H are evaluated on s_i. F is called the *head* of the static law (1) and the dynamic law (2).

The $C+$ language provides various abbreviations of the causal laws. The abbreviations that will be used in this paper are the following:

- We specify that fluent F is *inertial* as: **inertial** F. This is an abbreviation for a dynamic law of the form: **caused** F **if** F **after** F.
- The fact that action α cannot be executed if G holds is represented as: **nonexecutable** α **if** G which is an abbreviation of **caused** $\bot$ **after** $\alpha \wedge G$.
- The *closed-world assumption* regarding a formula F is represented as: **default** F which is an abbreviation of: **caused** F **if** F.

Any action description in $C+$ defines a *transition system*, a directed graph whose vertices are states and whose edges are labelled by actions. Given an action description D we can define the following:

- A *state* is an interpretation of σ^{rf} that satisfies $F \subset G$ 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. $T_{static}(s) = \{F \mid \text{static law (1) is in } D, s \models G\}$.
  2. $E(s, a, s') = \{F \mid \text{dynamic law (2) is in } D, s' \models G, s \cup a \models H\}$.
- A transition (s, a, s') is *causally explained* by D if and only if:
  1. $s' \models T_{static}(s')$.
  2. $s' \models E(s, a, s')$.
  3. There is no other s'' such that $s'' \models T_{static}(s')$ and $s'' \models E(s, a, s')$.

The *transition system* described by D is the following directed graph:

- The vertices are the *states* of D.
- 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 The Causal Calculator

The Causal Calculator (CCALC), a software system designed and implemented at the University of Texas[2], enables the representation of commonsense knowledge about action and change, and implements the $C+$ language that was described in the previous section [5]. CCALC has been applied to several challenge problems, e.g. [9]. Action descriptions in $C+$ are translated by CCALC first into the language of causal theories and then into propositional logic (via the process of completion of the definite causal theories) [5]. The models of the propositional theory correspond to paths in the transition system described by the original action description in $C+$. The input files of CCALC consist of comments, declarations and $C+$ causal laws regarding some action domain. These files are written in the input language of CCALC. The functionality of CCALC includes, among other things, computation of three kinds of tasks that can be represented as CCALC queries:

- *Prediction*. Given (partial or complete) information about an initial state and a complete sequence of actions, compute the information that holds in the resulting state (if there exists one) of a given transition system (action description) D.
- *Postdiction*. Given partial information about an initial state, (partial or complete) information about a resulting state, and, possibly, a (partial or complete) sequence of actions that leads from the initial state to the resulting one, compute some additional information that holds in the initial state (if there exists one) of a given transition system (action description) D.
- *Planning*. Given (partial or complete) information about an initial state and (partial or complete) information about a resulting state, compute the complete sequence of actions (if there exists one) that will lead from the initial state to the resulting one of a given transition system D.

In all of these computational tasks information (partial or complete) about intermediate states may be provided.

4 Specification of Open Electronic Societies

Artikis et al. [1] present a theoretical framework for providing executable specifications of *open computational societies*. A computational society is considered *open* (in [1]) if the following properties hold: First, the internal architecture of the agents is not publicly known. An open society can have members with different internal architectures. Therefore, open societies are treated as *heterogeneous* ones. Moreover, there is no direct access to an agent's mental state and so we can only infer things about it. Second, the members of the society do not necessarily have a notion of global utility [11]. Members may fail to, or choose not to, conform to specifications in order to achieve their individual goals. In addition

[2] http://www.cs.utexas.edu/users/tag/cc/

to these properties, in open societies 'the behaviour of the members and their interactions cannot be predicted in advance' [7].

In this framework the computational systems are viewed from an external perspective, that is to say, we are not concerned with the internal architecture of the agents. Three key components of computational systems are specified, namely the *social constraints*, *social roles* and *social states*. The specification of these concepts is based on and motivated by the formal study of legal and social systems (a theory of institutionalised power [8] and a theory of normative positions [12]) and traditional distributed computing techniques (state transition systems [4]). In [1] the social constraints and roles are specified by means of a subset of the 'full version' of the Event Calculus [14]. In this paper, we mainly focus on the social constraints and specify them with the use of the $C+$ language; a formalism with explicit state transition semantics. We illustrate the use of $C+$ by specifying a Contract-Net Protocol.

4.1 Social Constraints

We maintain, as in [1], the standard, in the study of social and legal systems, long established distinction between *permission*, *physical capability* and *institutionalised power* (see e.g. [8]). In other words, there is no standard (built-in) relationship between the actions that an agent is *physically capable* of performing, the actions that an agent is *permitted* to perform and the actions that an agent is *empowered* (by the institution/society) to perform. Accordingly, the social constraints specify the following:

- What kind of actions 'count as' [8] *valid* ('effective', 'meaningful') actions. Distinguishing between *valid* and *invalid actions* enables the separation of meaningful from meaningless activities.
- What kind of actions (valid, invalid) are *permitted*. Determining the *permitted*, *prohibited*, *obligatory actions* enables the classification of the agent behaviour as 'legal' or 'illegal', 'social' or 'anti-social', etc.
- What are the *sanctions* and *enforcement policies* that deal with 'illegal', 'anti-social' behaviour.

Valid actions are specified as follows: An action counts as a *valid action* at a point in time if and only if the agent that performed that action had the *institutionalised power* [8] to perform it at that point in time. Differentiating between valid ('meaningful') and invalid ('meaningless') 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.

On the second level of specification, the definition of *permitted*, *prohibited* or *obligatory* actions is *application-specific*. In other words, there is no standard definition of permitted actions — permitted actions may be defined in different ways in various computational societies. The same holds for the specification of sanctions; that is, the definition of sanctions may vary from one computational society to the other.

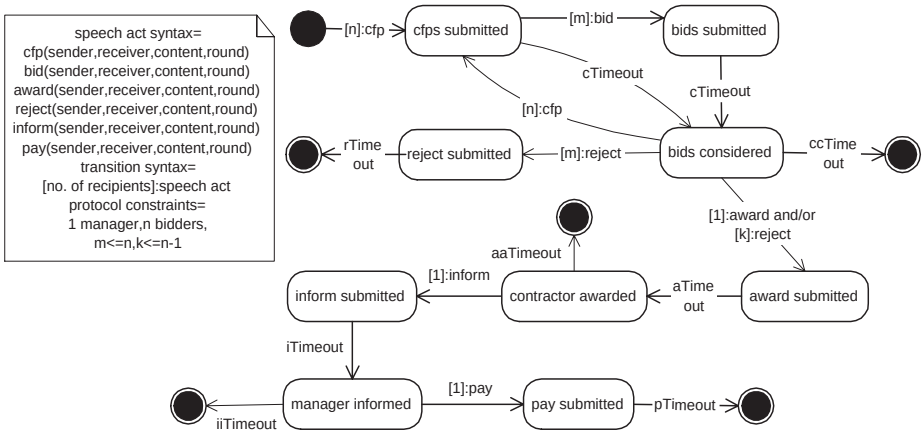

Fig. 1. The UML State Transition Diagram of the CNP [1].

5 A Contract-Net Protocol

In [10] we describe an abstract producer/consumer (APC) scenario where producers sell information to consumers. In this scenario the producers are explorer agents that map out the distribution of oil in their environment and consumers are cartographer agents that initiate Contract-Net Protocols to acquire the maps from the explorers. The main roles of the society that executes the Contract-Net Protocol (CNP) are that of the *cartographer* and that of the *explorer*. 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*[3]. 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.

Table 1 demonstrates a number of rigid, fluent and action constants and a causal law of the $C+$ representation of the CNP. Each of the constants will be described in the following sections. Two causal laws define when a state transition takes place. The first one states that in order to change state at least one action should take place (we employ an abbreviation provided by the input language of `CCALC` to represent this law):

$$\textbf{caused } \bot \textbf{ after } \neg d_1 \wedge \neg d_2 \wedge \ldots \wedge \neg d_n \text{ for every action constant } d_i \quad (3)$$

[3] The protocol round is a parameter of the description of the actions that agents perform (i.e. action constant). 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*.

Table 1. A partial $C+$ description of the CNP.

Notation:
roleName ranges over {*Cartographer*, *Explorer*}
agent,*agent2*,*e*,*c* range over {$Cartographer_1, Explorer_1, \ldots, Explorer_n$}
perf ranges over {*Cfp*, *Bid*, *Award*, *Reject*, *Inform*, *Pay*}
content,*content2* range over a finite set of task descriptions
round ranges over $\mathbb{N}^+$, the set of positive integers
Rigid constant (domain is {*Cartographer*, *Explorer*}):
$Role\_of(agent)$
Simple fluent constant (boolean domain):
$ValidActionHappened(agent, perf, agent, content, round)$
Statically determined fluent constants (boolean domain):
$Pow(agent, perf, agent, content, round)$,
$Permitted(agent, perf, agent, content, round)$,
$Obliged(agent, perf, agent, content, round)$,
$Sanction(agent)$
Action constant (boolean domain):
$Valid(agent, perf, agent, content, round)$
Causal laws:
inertial c for every simple fluent constant c

The second one states that at most one action can take place at each time:

noconcurrency (4)

The result of these two laws is that in order to change state exactly one action should take place. Such a restriction can be lifted in different specifications of the CNP. We introduce this restriction just to simplify our presentation. A number of the remaining causal laws of the formalisation of the CNP are described in the following sections.

5.1 Social Constraints

Valid Actions The actions that the agents perform can be classified as *valid* or *invalid*. An action will count as [8] a valid one if and only if the agent that performed that action had the institutional power to perform that action. For example, the specification of valid actions could be represented by a static law

of the form:

$$\begin{aligned} \textbf{caused}\ & \mathit{Valid}(\mathit{agent}, \mathit{perf}, \mathit{agent2}, \mathit{content}, \mathit{round})\ \textbf{if} \\ & \mathit{Pow}(\mathit{agent}, \mathit{perf}, \mathit{agent2}, \mathit{content}, \mathit{round}) \wedge \\ & \mathit{Action}(\mathit{agent}, \mathit{perf}, \mathit{agent2}, \mathit{content}, \mathit{round}) \end{aligned} \tag{5}$$

In (5) *Valid* is an action constant representing valid actions, *Action* is an action constant representing the agents' actions and *Pow* is a fluent constant representing the institutional powers of the agents. Constraint (5) states that if an *agent* performs an action (represented by *Action*) and he has the institutionalised power to perform that action (represented by *Pow*) then this action is considered valid. However, we cannot represent such a constraint in the $C+$ language, since it is not possible to have an action constant both in the head and the body of a causal law[4]. Since it is not possible to specify such a constraint in $C+$, we have defined the following constraint (in order to approximate constraint (5)):

$$\begin{aligned} \textbf{nonexecutable}\ & \mathit{Valid}(\mathit{agent}, \mathit{perf}, \mathit{agent2}, \mathit{content}, \mathit{round})\ \textbf{if} \\ & \neg \mathit{Pow}(\mathit{agent}, \mathit{perf}, \mathit{agent2}, \mathit{content}, \mathit{round}) \end{aligned} \tag{6}$$

According to constraint (6) a *Valid* action is **nonexecutable** if the agent that performs it does not have the institutionalised power to do so. In other words, an agent can perform a valid action only if he has the power to do so.

In the applications we have in mind we want to keep track of the valid actions as they occur. Therefore, we use *ValidActionHappened*, a *simple fluent constant* to record the fact that a valid action has happened. We are aware that encoding fluents (that represent the history) in a state description in this manner runs counter to the spirit of transition systems. We are currently examining alternative formalisations that avoid this issue.

Since valid actions are determined in terms of institutional powers, it is important to specify these powers. We represent the institutional powers with the use of a statically determined fluent. There are several rules that define ('statically determine') the *Pow* fluents. For example, the power to *Bid* is defined as:

$$\begin{aligned} \textbf{caused}\ & \mathit{Pow}(e, \mathit{Bid}, c, \mathit{content}, \mathit{round})\ \textbf{if} \\ & \mathit{ValidActionHappened}(c, \mathit{Cfp}, e, \mathit{content}, \mathit{round}) \wedge \\ & \neg \mathit{CTimeoutHappened}(\mathit{round}) \wedge \\ & \forall \mathit{content2} : \neg \mathit{ValidActionHappened}(e, \mathit{Bid}, c, \mathit{content2}, \mathit{round}) \end{aligned} \tag{7}$$

The above law specifies that an explorer (represented by the e variable) has the power to *Bid* if the following conditions hold:

- A cartographer (represented by the c variable) has performed a *Valid Cfp*.
- A *CTimeout* has *not* taken place.

[4] Such a constraint can be represented in the extended $C+$ [13] (see Section 8).

– The explorer has *not* performed a *Valid Bid*.

Only when these conditions hold, will an explorer be empowered to perform a *Bid*. Such a *closed-world assumption* regarding the specification of powers is defined as follows:

$$\textbf{default } \neg Pow(agent, perf, agent2, content, round) \tag{8}$$

In other words, in the absence of information to the contrary (e.g. constraint (7)), no agent is empowered to perform an action.

Apart from the actions that agents perform, the specification of the CNP includes a number of additional actions/events, the *timeout events*. These events are *system events*, in the sense that they are performed by a global clock, and are considered valid. The timeout events are represented by a number of action constants. Due to space limitations, we have omitted the representation of these constants and the laws that are associated with them from the presented analysis.

Permitted Actions We now specify what actions are *permitted* during the execution of the CNP. As for the *Pow* fluent, we represent permitted actions with the use of a statically determined fluent, the *Permitted* fluent. For this variation of the CNP we have specified that an agent is permitted to perform an action *if and only if* he has the power to perform that action:

$$\textbf{default } \neg Permitted(agent, perf, agent2, content, round) \tag{9}$$

$$\textbf{caused } Permitted(agent, perf, agent2, content, round) \textbf{ if } Pow(agent, perf, agent2, content, round) \tag{10}$$

The specification of the permitted actions is *application-specific*. In different settings we might specify permissions in a different manner. For example, we might forbid agents in certain circumstances to perform actions even if they are empowered to do so.

Apart from the valid and permitted actions, the specification of the CNP includes definitions of *obligations*, *sanctions* and *social roles*. We represent obligations and sanctions with the use of *statically determined fluent constants* (as in the case of powers and permissions). We represent social roles [1] with *rigid constants* (see Table 1) on the assumption that the agents do not change their roles (i.e. *Cartographer*, *Explorer*) during the execution of the CNP. Of course, such a restriction can be lifted in different specifications of the CNP. Due to space limitations, we omit the presentation of the causal laws that are associated with obligatory actions, sanctions and social roles.

5.2 Social States

The *action signature* of the formalisation of the CNP can be specified as:

– $\sigma^{rf} \supseteq \{ValidActionHappened, Pow, Permitted\}$. σ^{rf} is the set of rigid and fluent constants.

– $\sigma^{act} \supseteq \{Valid\}$. σ^{act} is the set of action constants[5].

The action description D of the CNP is a set of causal laws, some of which were presented in the previous sections of this paper. This action description defines a transition system where:

– The vertices are *states* of D, i.e. interpretations of σ^{rf}.
– The edges (s, a, s') are *causally explained* by D.

Consider a CNP with only two agents, cartographer C and explorer E, and the transition (s, a, s') where:

– $s \supseteq \{Pow(C, Cfp, E, MapUK, 1), Permitted(C, Cfp, E, MapUK, 1)\}$.
– $a = \{Valid(C, Cfp, E, MapUK, 1)\}$.
– $s' \supseteq \{Pow(E, Bid, C, MapUK, 1), \neg Permitted(E, Bid, C, MapUK, 1),$
$ValidActionHappened(C, Cfp, E, MapUK, 1)\}$.

In order to determine whether this transition is an edge of the transition system defined by D we need to determine if this transition is *causally explained* by D (see Section 2). Intuitively, constraints (9) and (10) specify (in this example) that we cannot have a state where an agent is empowered, but not permitted, to perform an action. Because of constraint (10), $T_{static}(s') \supseteq \{Permitted(E, Bid, C, MapUK, 1)\}$. Consequently, $s' \not\models T_{static}(s')$ and (s, a, s') is not causally explained by D. (s, a, s') is not an edge of the transition system defined by D. Similarly, we can determine which state transitions *are* edges of the transition system defined by D.

6 Executing the Specifications

We used the $C+$ language to declare the constants (Table 1) and specify the laws of the formalisation of the CNP. The $C+$ formalisation is translated to the input language of the `Causal Calculator` (`CCALC`) in order perform a number of computational experiments with our formalisation of the CNP. For example, constraint (6) is written in the following way in the input language of `CCALC`[6]:

```
nonexecutable valid(Agent, Perf, Agent2, Content, Round) if
              -pow(Agent, Perf, Agent2, Content, Round).
```

To test our formalisation of the CNP we performed a number of queries to `CCALC`. The tested formalisation includes four agents; `cartographer1` occupies the role of the `cartographer` and `explorer1`, `explorer2`, `explorer3` occupy the role of `explorer`. The task description is `mapUK` and the protocol can have at most two rounds (`1..2`).

(Prediction) Query 1. We are in the *bids considered state* (see Figure 1). The following events have taken place: all of the explorers have performed `valid`

[5] For clarity reasons we omit the parameters of the constants.
[6] See [5, 9] for details about the syntax of the input language of `CCALC`.

bids about mapUK (the protocol round is 1) and the first timeout (i.e. cTimeout) has elapsed. cartographer1 is empowered to perform a new cfp or to award or reject the three bids he has received. In this state, can cartographer1 perform a valid reject to explorer1 regarding mapUK at the first protocol round? If yes, what are the new powers associated with cartographer1?
CCALC determines that the valid(cartographer1,reject,explorer1,mapUK,1) action is executable. The resulting state includes the following powers:

```
1: pow(cartographer1, award, explorer2, mapUK, 1);
   pow(cartographer1, award, explorer3, mapUK, 1);
   pow(cartographer1, reject, explorer2, mapUK, 1);
   pow(cartographer1, reject, explorer3, mapUK, 1).
```

(Planning) Query 2. Given the initial state of the CNP, i.e. cartographer1 is empowered to issue a cfp to the three explorers, is it possible to find a state (within 16 steps[7]) where some explorer has been awarded some task and cartographer1 is permitted to award some other explorer? This question can be represented by the following CCALC query:

```
:- query
maxstep :: 1..16;
0: pow(cartographer1, cfp, explorer1, mapUK, 1),
   pow(cartographer1, cfp ,explorer2, mapUK, 1),
   pow(cartographer1, cfp, explorer3, mapUK, 1);
maxstep:
[\/Content \/Agent \/Round |
   validActionHappened(cartographer1,award,Agent,Content,Round)] &
[\/Content2 \/Agent2 \/Round2|
   permitted(cartographer1,award,Agent2,Content2,Round2)].
```

CCALC finds no solution within 16 steps. Due to constraints (3) and (4), the maximum number of steps in this CNP is 16. Since there is no solution within 16 steps (starting from the initial state), the following statement holds: In this specification of the CNP it is not possible to reach a state where the agent occupying the role of the cartographer has awarded an agent and is permitted to award another agent.

(Postdiction) Query 3. Initially, cartographer1 was permitted to issue a cfp to explorer3 regarding mapUK at the first protocol round. After one step cartographer1 is empowered to award explorer1 regarding mapUK at the first protocol round. What can we say initially about the powers of cartographer1?

CCALC finds several solutions. The action (that leads from the initial state to the resulting one) at each solution was cTimeout, i.e. the timeout after a valid cfp. In all of these solutions, the initial state included the following:

```
0: permitted(cartographer1, cfp, explorer3, mapUK, 1);
```

[7] A *step* is a transition from one state to the next. Due to constraints (3), (4), in this specification of the CNP a step takes place when exactly one action takes place.

```
validActionHappened(explorer1, bid, cartographer1, mapUK, 1);
pow(cartographer1, cfp, explorer3, mapUK, 1).
```

6.1 Evaluation

In our experiments (four agents, two task descriptions, two protocol rounds) CCALC generated big theories: over 2000 atoms, over 20000 rules and over 30000 clauses. Prediction queries were computed in 45 seconds on a Pentium IV 2GHz, 1GB RAM computer. Increasing the number of agents, task descriptions or protocol rounds (scalability of the formalisation) leads to larger theories that make CCALC compute queries in a less timely fashion.

Based on our formalisation of the CNP and its execution in CCALC, we have reached the conclusion that CCALC does not seem suitable for on-line activities, that is, activities during the actual execution of the societies. On-line activities include, among other things, the compilation of the social states (i.e. what powers, permissions, obligations and sanctions are associated with each member of the society at each time point) during the simulations of electronic societies. The compilation of the social states is performed with the use of prediction queries. The computation of prediction queries, as shown above, is not performed sufficiently fast for on-line activities.

As already mentioned, in previous work [1] we formalised the CNP with the use of a sub-set of the 'full version' of the Event Calculus [14] and implemented this formalisation in the Prolog programming language. This implementation, a software tool called the *Society Visualiser* (SV), can compute prediction queries regarding the Event Calculus specification of the CNP. Given the same experimental settings, the Society Visualiser computed prediction queries sufficiently fast for on-line activities.

Unlike CCALC, the Society Visualiser cannot compute planning and postdiction queries. This limitation of the SV can be lifted by employing an Event Calculus planner [14] (see Section 7 for a use of such a planner in the context of specification of interaction protocols). This is an issue that we have not yet addressed.

CCALC can be used in various settings/configurations with respect to the simulation/actual execution of computational societies. One possible setting is the following: CCALC acting as a central entity in a computational society, monitoring (or even auditing) the execution of the members of the society and producing the social states of the society. In this setting, members of the computational systems can query CCALC in order to determine the powers and normative positions relevant to them and their peers. In a different setting, the functionality of CCALC may be distributed. For example, each member of the computational system may have a module with similar computational capabilitites as CCALC. This module will produce the powers and normative positions of the agent that it belongs to.

7 Related Work

There are several approaches in the DAI literature that come close to the objectives of our work. A few notable examples are work on *e-institutions* [3], work on *commitment protocols* [19] and work on *negotiation protocols* [2].

Yolum and Singh [19] present a formalisation of the *commitment protocols* [16] in terms of the *Event Calculus*. Moreover, they employ an abductive Event Calculus planner [14] in order facilitate the planning of the agents that execute these protocols. Our work is very similar to [19]. We employ `CCALC` to perform planning as well as prediction and postdiction queries regarding the specifications of computational systems that are formalised in $C+$.

Bartolini and colleagues [2] focus on the specification of *negotiation mechanisms* (rather than *negotiation strategies*) and argue that the *negotiation rules* should be made explicit at the design stage of a multi-agent system rather than being implicitly specified in the minds of the participants of the negotiation protocols. Furthermore, they propose a software framework for automated negotiation that provides several functionalities to the participants of the negotiation protocols. For example, agents can access the negotiation rules at any time in order to determine/modify their strategies.

Our work has similaritites with the work of Bartolini et al. [2]. Negotiation protocols can be viewed as types of 'open computational societies' (as these have been defined in this paper). There is no access to the internals of the participants of the negotiation protocols. Moreover, the participants are negotiating in order to achieve their antagonistic goals. Like [2], our motivation is that the rules (i.e. social constraints) of such computational systems should be explicitly (and formally) defined at the design stage of the computational systems. The software framework for automated negotiation of [2] bears some similarities with the way we use `CCALC` for the execution of the specifications of the computational systems. In particular, *negotiation hosts* [2] provide similar functionalities to the ones that `CCALC` provides (especially when `CCALC` is used as a central entity — see Section 6.1).

A key difference between our work and the work reviewed here is that we explicitly represent the institutional powers of the members of the computational systems and differentiate between institutional power, physical capability and permission. Jones and Sergot [8] pointed out that " 'empowering' is not an *exclusively* legal phenomenon, but is a standard feature of any norm-governed organisation where selected agents are assigned to specific roles (in which they are empowered to conduct the business of that organisation)". The concept of *institutionalised power*, although being a standard feature of any norm-governed institution/society, is not explicitly represented in the reviewed approaches. Singh [15] provides an implicit representation of the concept of institutional power in his work on commitment protocols. Moreover, the reviewed approaches do not differentiate between institutional power, permission and physical power.

8 Summary and Current Work

The functionality of the `Causal Calculator` (`CCALC`) regarding the specification of open electronic societies can be summarised as follows:

- `CCALC` implements $C+$, a language with explicit state transition semantics, support for effects (direct and indirect) of actions and default ('inertia') persistence of fluents from state to state. As shown in Sections 2 and 5.2 an action description in $C+$ defines a transition system of a particular kind.
- `CCALC` provides a tool that enables the society designers and the agent designers to 'validate' the specifications of the social laws after the design stage. The validation is mainly performed with planning and postdiction queries. These type of queries enable the designers to prove various properties of the protocols/society specifications (e.g. see Query 2 in Section 6). In this way, agent designers can perform planning queries in an off-line phase (i.e. before the commencement of the actual execution of the societies) in order to determine whether it is desirable or not to deploy their agents in societies.
- `CCALC` does not seem suitable for on-line activities (i.e. activities during the actual execution of the societies) such as the compilation of the social states (i.e. what powers, permissions, obligations, sanctions, roles are associated with each member of the society at each time point) during the simulations of electronic societies.

Current work includes two main directions. Sergot [13] presents an extension of the $C+$ language that includes direct support for specification of (a version of) the 'counts as' relation for action [8] and a treatment of permitted/forbidden states, actions and paths (in the spirit of [18]). We are currently formalising our framework using extended $C+$ in order to directly use laws of the form (5) and not approximate them with laws of the form (6).

Furthermore, we are investigating ways of improving the computation of `CCALC` rules resulting from the society specification shown. More specifically, we aim to improve the way `CCALC` computes prediction queries. In order to do that, we aim to identify and discard the $C+$ laws that are not relevant to the (prediction) query in question. In this way, `CCALC` will have as input a smaller number of $C+$ laws and will therefore generate a smaller number of rules and clauses (via the process of completion of the definite causal theories [5]).

It is important to note that `CCALC` is a tool under development and, therefore, some of these issues (large number of atoms and scalability) may be addressed in future versions[8].

9 Acknowledgements

This work has been undertaken in the context of the EU-funded ALFEBIITE Project (IST-1999-10298). We would also like to thank Vladimir Lifschitz and Joohyung Lee from the Action Group at the University of Texas for their suggestions regarding the $C+$ language and `CCALC`.

[8] We used `ccalc 2.04b` for the computational experiments.

References

1. A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In C. Castelfranchi and L. Johnson, editors, *Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1053–1062, 2002.
2. C. Bartolini, C. Priest, and N. Jennings. Architecting for reuse: A software framework for automated negotiation. In *Proceedings of Workshop on Agent-Oriented Software Engineering (AOSE)*, pages 87–98, 2002.
3. M. Esteva, J. Rodriguez-Aguilar, C. Sierra, P. Garcia, and J. Arcos. On the formal specifications of electronic institutions. In F. Dignum and C. Sierra, editors, *Agent Mediated Electronic Commerce*, LNAI 1991, pages 126–147. Springer, 2001.
4. R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. The MIT Press, 1995.
5. E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. 2001.
6. M. Hardwick and R. Bolton. The industrial virtual enterprise. *Communications of the ACM*, 40(9):59–60, 1997.
7. C. Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47:76–106, 1991.
8. A. Jones and M. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3), 1996.
9. J. Lee, V. Lifschitz, and H. Turner. A representation of the zoo world in the language of the causal calculator. In *Proceedings of Fifth Symposium on Formalizations of Commonsense Knowledge*, 2001.
10. J. Pitt, L. Kamara, and A. Artikis. Interaction patterns and observable commitments in a multi-agent trading scenario. In *Proceedings of Conference on Autonomous Agents (AA)*, pages 481–489. ACM Press, 2001.
11. J. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. The MIT Press, 1998.
12. M. Sergot. A computational theory of normative positions. *ACM Transactions on Computational Logic*, 2(4):522–581, 2001.
13. M. Sergot. The language $(\mathcal{C}/\mathcal{C}+)^{++}$. *ALFEBIITE Deliverable D6(2)*, 2002.
14. M. Shanahan. The event calculus explained. *Artificial Intelligence Today*, pages 409–430, 1999.
15. M. Singh. An ontology for commitments in multiagent systems: Towards a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.
16. M. Singh. A social semantics for agent communication languages. In *Issues in Agent Communication*, LNCS 1916, pages 31–45. Springer, 2000.
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. R. van der Meyden. The dynamic logic of permission. *Journal of Logic and Computation*, 6:465–479, 1996.
19. 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 Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 527–535, 2002.