Specifying and Reasoning about Multiple Institutions

Owen Cliffe, Marina De Vos, and Julian Padget

Department of Computer Science
University of Bath, BATH BA2 7AY, UK
{occ,mdv,jap}@cs.bath.ac.uk

Abstract. Correctly specifying the behaviour of normative systems such as contracts and institutions is a troublesome problem. Designers are faced with two concurrent, difficult tasks: firstly specifying the relationships (over time) of agents' actions and their effects, and secondly combining this model with another that captures the agents' permissions and obligations. In this paper we present our model and operational semantics for specifying individual and collective institutions and outline a declarative action language for describing them. We demonstrate, by way of an example, how this may be used to enable the analysis of institutional specifications either for simply visualising possible outcomes or for checking for absence or presence of certain (un)desirable correctness properties.

1 Introduction

Institutions have long been studied in the multi agent systems community as a means for capturing the *social semantics* of interactions among agents. While a lot of work [15, 18, 19, 17, 7, 16, 1, 5, 5] has focused on modelling single institutions, nobody so far has addressed the issue of modelling multiple interacting institutions. In this case, particular aspects of a society may be modelled individually and then combined to give a richer model, leading to the possibility of using institutions as a means for abstraction (capturing increasing levels of specifi city at lower levels) and also as a means for delegation (whereby one institution relies on the behaviour of another to augment its function). A good example is a contract violation which is considered as a breach of civil law. The contract and the civil law are themselves independent entities, and one could argue that a formal contract exists without the force of law, however the presence of this institution leads to more force behind the contract—that is, a victim of contract violation may have a reasonable expectation that the violator will be sanctioned elsewhere.

Action languages have evolved over recent years as a means of providing declarative, human-readable descriptions of the effects of actions and events. In [10] Gelfond and Lifschitz summarise action languages thus: "*Abstract Action languages are formal models of parts of the natural language that are used for talking about the effects of actions.*" The semantics of action languages are typically described over a transition system where each state (or situation) is composed of the valuations of zero or more fluents and each transition is modelled by one or more action symbols.

Action languages are typically used for the analysis of situations and in this case, the action language describes a model of the situation which can then be queried to determine various properties. An intuitive and elegant way of doing this is to map the action

language to an answer set program. Answer set programming (ASP) is a logic programming language that admits reasoning about possible world views in the absence of complete information. Due to its formal semantics, and combined with efficient heuristic solvers, answer set programming, provides an excellent basis from which derived models may be queried. More information about answer set programming and its applications can be found in [3].

In this paper, we extend the formal specification of single institutions in [4] to multi-institutions. We present a top-down approach to virtual multi-institutions, in which external normative concepts are represented in forms that at the same time designers may analyse (off-line) and about which agents may reason (on-line). Instead of using ASP directly (as in [5, 4]), we introduce an action language designed for multi-institutions. The use of the action language makes generating the ASP code less open to human coding error, and perhaps more importantly easier to understand and create without losing either expressiveness or a formal basis for the language by narrowing the semantic gap.

2 Multi-Institutions

2.1 The Single Institution

To provide some context for the theory that follows, this section begins with a brief overview of institutions and the terms that we use. As outlined in the introduction the essential characteristics of an institution are captured in its norms with varying degrees of specificity. What agents do or say is constrained by the institutional context, so that irrelevant actions or communications are ignored, and relevant ones advance the interaction, cause an agent to acquire an obligation, or through a violation, invite a sanction. But while that serves to capture the agent's point of view, what about the (institutional) environment? How are actions to be observed, how are obligations to be recorded and their satisfaction enforced, and how are violations to be detected and the corresponding sanctions to be applied?

The model we propose is based on the concept of *Observable Events* that capture notions of the physical world — "shoot somebody" — and *Institutional Events* that are those generated by society — "murder" — but which only have meaning within a given social context. While observable events are clearly observable, institutional ones are not, so how do they come into being? Searle [11] describes the creation of an institutional state of affairs through *Conventional Generation*, whereby an event in one context *Counts As* the occurrence of another event in a second context. Taking the physical world as the first context and by defining conditions in terms of states, institutional events may be created that count as the presence of states or the occurrence of events in the institutional world.

Thus, we model an institution as a set of *institutional states* that evolve over time subject to the occurrence of *events*, where an institutional state is a set of *institutional fluents* that may be held to be true at some instant. Furthermore, we may separate such fluents into *domain* fluents, that depend on the institution being modelled, such as "A owns something", and *normative fluents* that are common to all specifications and may be classified as follows:

– **Institutional Power:** This represents the institutional capability for an event to be brought about meaningfully, and hence change some fluents in the institutional

state. Without institutional power, the event may not be brought about and has no effect; for example, a marriage ceremony will only bring about the married state, if the person performing the ceremony is empowered so to do.

– **Permission:** Each permission fluent captures the property that some event may occur without violation. If an event occurs, and that event is not permitted, then a *violation event* is generated.
– **Obligation:** Obligation fluents are modelled as the dual of permission. An obligation fluent states that a particular event is obliged to occur before a given deadline event (such as a timeout) and is associated with a specified violation. If an obligation fluent holds and the obliged event occurs then the obligation is said to be satisfied. If the corresponding deadline event occurs then the obligation is said to be violated and the specified violation event is generated.

Events can be classified into: (i) a set of observable events, being those events external to the institution which may be brought about independently from the institution and (ii) a set of *institutional events* which may be broken down into *violation events* and *institutional actions*; these events may only be brought about if they are generated by the institutional semantics. Finally we have a set of institutional rules which associate the occurrence of events with some effects in the subsequent state. These can be divided into: (i) *generation rules* which account for the conventional generation of events. Each generation rule associates the satisfaction of some conditions in the current institutional state and the occurrence of an (observed or institutional) event with a generated institutional event. For example: "A wedding ceremony counts as civil marriage only if the couple have a licence". The generating and generated events are taken by the institution to have occured simultaneously. (ii) *consequence rules*, each of which associates the satisfaction of some conditions in the current institutional state and the occurrence of an event in the institution or the world to the change in state of one or more fluents in the next institution state. For example: "Submitting a paper to a conference grants permission for the paper to be redistributed by the conference organisers".

Violation and sanction play an important role in the specification of institutions. Violations may arise either from explicit generation, from the occurrence of a non-permitted event, or from the failure to fulfil an obligation. In these cases sanctions that may include obligations on violating agents or other agents and/or changes in agents' permission to do certain actions, may then simply be expressed as consequences of the occurrence of the associated violation event in the subsequent institutional state.

2.2 Combining Institutions

Institutions are not necessarily separate entities; several of them could operate within the same context, agents can participate in a number of them at the same time and perhaps more interestingly institutions themselves can be governed by institutions. In this section we investigate how such relationships can be established. To our knowledge we are the first to examine this topic.

Just as agents have the right to join in or stay out of an institution, it should be up to the institution to allow other institiutions to change directly or indirectly its state. In other words, it needs to put in place a mechanism of empowerment that allows institutions to bring about events and to initiate and terminate fluents within the institution.

Providing other institutions with the power to bring about certain events within the institution can easily be supported using the existing single institutional framework by quantifying the institutional power with the institution that is given the power. Since verifying empowerment is already part of event generation for a single institution, we do not need to change event generation in the presence of multiple institutions.

Things are different, when we want to empower institutions to change directly each others' state, because as for single institutions, empowerment only ranges over events. In the presence of multi-instititutions, we need to introduce two new empowerments: one for initialising and one for terminating fluents. These two institutional powers will then be used in conjunctions with the consequence rules to determine the next state of the institution.

2.3 Operational Specification

The model

Each multi-institution specification $\mathcal{M}$ is characterised by the institutions $\mathcal{I}$ that constitute it. Thus, $\mathcal{M} = \langle \mathcal{I}_1, \ldots, \mathcal{I}_n \rangle$ is a sequence of individual institutions $\mathcal{I}_i$. Each of these institutions is a five-tuple $\mathcal{I}_i := \langle \mathcal{E}_i, \mathcal{F}_i, \mathcal{C}_i, \mathcal{G}_i, \Delta_i \rangle$ with institutional Events ($\mathcal{E}_i$), Fluents ($\mathcal{F}_i$), Consequences ($\mathcal{C}_i$), Event Generation ($\mathcal{G}_i$) and Initial State (Δ_i). In the following subsections we discuss the various parts in more detail and their effect on the multi-institution $\mathcal{M}$.

Institutional Events Each institution $\mathcal{I}_i$ defines a set of event signatures $e \in \mathcal{E}_i$, to denote the types of event that may occur. $\mathcal{E}_i$ comprises two disjoint subsets, $\mathcal{E}^i_{obs}$ denoting *observable events* and $\mathcal{E}^i_{inst}$ denoting *institutional events*. We break institutional events down further into the disjoint subsets: *institutional actions* $\mathcal{E}^i_{instact}$ and *violation events* $\mathcal{E}^i_{viol}$. We define $\mathcal{E}^i_{viol}$ such that $\forall e \in \mathcal{E}^i_{instact} \cdot \mathrm{viol}(e) \in \mathcal{E}^i_{viol}$: that is each institutional action has a corresponding violation event $\mathrm{viol}(e)$ in $\mathcal{E}^i_{viol}$ which may arise from performing e when it is not permitted. Other violations can be added to indicate agents have not behaved as they should have, e.g. fulfilling an obligation.

We assume that the individual institutions have disjoint sets of events $\mathcal{E}_i$[1]. We define the set of all events of the multi-institution $\mathcal{M}$, by $\mathcal{E}_{\mathcal{M}} = \bigcup_{i=1}^{n} \mathcal{E}_i$. We define $\mathcal{E}^{\mathcal{M}}_{inst}$, $\mathcal{E}^{\mathcal{M}}_{obs}$, $\mathcal{E}^{\mathcal{M}}_{instact}$ and $\mathcal{E}^{\mathcal{M}}_{viol}$ in a similar fashion. To obtain the corresponding institution for any event e, we define the function $\rho : \mathcal{E}_{\mathcal{M}} \to \mathbb{N}$ such that $\rho(e) = i$ when $e \in \mathcal{E}_i$.

Institutional Fluents Each institution $\mathcal{I}_i$ defines a set of *Domain Fluents* denoted $\mathcal{D}_i$ which is a set of fluents modelling the context in which the institution is operational. In addition to the domain fluents, we define a number of disjoint sets of boolean fluents, $\mathcal{W}_i$, $\mathcal{P}_i$, $\mathcal{O}_i$, $\mathcal{S}_i$ and $\mathcal{T}_i$, indicating different types of *normative fluents*. Together, these disjoint sets of domain fluents and normative fluents form the *Institutional Fluents* $\mathcal{F}_i$ ($\mathcal{F}_i = \mathcal{W}_i \cup \mathcal{P}_i \cup \mathcal{O}_i \cup \mathcal{S}_i \cup \mathcal{T}_i \cup \mathcal{D}_i$). The set of all available fluents in the multi-institution $\mathcal{M}$ is denoted as $\mathcal{F}_{\mathcal{M}}$ ($\mathcal{F}_{\mathcal{M}} = \bigcup_{j=1}^{n} \mathcal{F}_j$).

$\mathcal{W}_i$ A set of institutional powers of the form $\mathrm{pow}(j, e) : 1 \leq j \leq n, e \in \mathcal{E}^i_{instact}$ where each power fluent denotes the capability of some event e to be brought about in the institution.

[1] This may seem as a limitation, especially when the observable events are conscerned, but it is not. We are modelling events from the viewpoint of an institution and not from the events themselves.

$\mathcal{P}_i$ A set of action permissions: $\mathrm{perm}(e) : e \in \mathcal{E}^i_{instact}$ where each permission fluent denotes that it is permitted for action e to be brought about. An event is not explicitly forbidden, instead this is implicitly represented through the absence of permission for that event to occur.

$\mathcal{O}_i$ A set of obligations, of the form $\mathrm{obl}(e, d, v) : e \in \mathcal{E}_i, d \in \mathcal{E}_i, v \in \mathcal{E}^i_{inst}$ where each obligation fluent denotes that action e should be brought about before the occurrence of event d or be subject to the violation v. Note that v need not necessarily be a violation, but any event which represents the failure to satisfy the obligation.

$\mathcal{S}_i$ A set of institutional initiating powers of the form $\mathrm{inipow}(j, f)$: $1 \leq j \leq n$, where $f \in \mathcal{D}_i$ denotes that institution j is empowered to initiate some domain fluent f in institution i.

$\mathcal{T}_i$ A set of institutional terminating powers of the form $\mathrm{termpow}(j, f)$: $1 \leq j \leq n$, where $f \in \mathcal{D}_i$ denotes that institution j is empowered to terminate some fluent f in institution i.

The state of an institution at a certain time is determined by those institutional fluents that are valid at that time. So a state S is a subset of $\mathcal{F}$. A fluent f which is not valid is denoted as $\neg f$. This notation can be extended to sets of fluents.

The set of all possible *institutional states* of institution $\mathcal{I}_i$ is denoted as Σ_i with $\Sigma_i = 2^{\mathcal{F}_i}$. It is important to note that not all those states will actually be reachable in an institution. The state of the multi-institution $\mathcal{M}$ is modelled as a sequence $\langle S_1, \ldots, S_n \rangle$ with $S_i \in \Sigma_i$. The state of all possible states for $\mathcal{M}$ is defi ned as $\Sigma_{\mathcal{M}} = \Sigma_1 \times \ldots \times \Sigma_n$.

Events can have the same effect on a number of states. Borrowing a book from a library will result in the obligation to bring it back regardless of how many books are currently on loan. To facilitate this, we introduce the concept of *State Formulae* to capture a collection of states that satisfy certain properties in that they either contain certain fluents or they do not. The set of all state formulae is denoted as $\mathcal{X}_i$ with $\mathcal{X}_i = 2^{\mathcal{F}_i \cup \neg\mathcal{F}_i}$, where $\neg\mathcal{F}_i$ is the negation of each fluent in $\mathcal{F}_i$.

Consequences Each institution $\mathcal{I}_i$ defi nes the function $\mathcal{C}_i$ which describes which fluents are initiated and terminated by the occurrence of a certain event in a state matching some criteria. The function is expressed as $\mathcal{C}_i : \mathcal{X}_i \times \mathcal{E}_i \rightarrow 2^{\mathcal{F}_{\mathcal{M}}} \times 2^{\mathcal{F}_{\mathcal{M}}}$. Given $X \in \mathcal{X}_i$ and $e \in events_i$, $C_i(X, e) = (\mathcal{C}^{\uparrow}_i(X, e), \mathcal{C}^{\downarrow}_i(X, e))$ with $\mathcal{C}^{\uparrow}_i(X, e)$ containing those fluents which are initiated by the event e in a state matching X and $\mathcal{C}^{\downarrow}_i(X, e)$ collecting those fluents which are terminated by event e in a state matching X. Notice that the consequence relation can indicate which events can cause fluents to change in the state of institutions different from itself. This will only take effect if the institution is empowered to do so.

Event Generation Each institution $\mathcal{I}_i$ defi nes an event generation function $\mathcal{G}_i$ which describes when the occurrence of one event *counts as* the occurrence of another:
$\mathcal{G}_i : \mathcal{X}_i \times \mathcal{E}_i \rightarrow 2^{\mathcal{E}^{\mathcal{M}}_{inst}}$.

Initial State Each institution $\mathcal{I}_i$ defi nes the set $\Delta_i \subseteq \mathcal{F}_i$ which denotes the set of fluents that should hold when the institution is created.

The initial state of the multi-institution $\mathcal{M}$ is the sequence $\Delta_{\mathcal{M}} = \langle \Delta_1, \ldots, \Delta_n \rangle$.

2.4 Semantics

During the lifetime of an institution, its state changes due to events that take place. Each observable event possibly generates more events which in turn could create further

events. Each of these events could have an effect on the current state. The combined effect of these events determines the next state.

States We define the semantics of a multi-institution $\mathcal{M}$ over a set of states $\Sigma_{\mathcal{M}}$. Each $S \in \Sigma_{\mathcal{M}}$ consists of a sequence containing a state $S_i \in \Sigma_i$ for each institution $\mathcal{I}_i$ in $\mathcal{M}$. Each state S_i is a set of fluents in $\mathcal{F}_i$ which are held to be true at a given time. We say that $S \in \Sigma_{\mathcal{M}}$ satisfies fluent $f \in \mathcal{F}_i$, denoted $S \models f$, when $f \in S_i$. It satisfies its negation $\neg f$, when $f \notin S_i$. This notation can be extended to sets $X \subseteq \mathcal{X}_i$ in the following way: $S \models X$ iff $\forall x \in X \cdot S \models x$.

Event Generation In order to model event generation we define function which describes which events are generated in a given state. $\mathrm{GR} : \Sigma_{\mathcal{M}} \times 2^{\mathcal{E}_{\mathcal{M}}} \rightarrow \mathcal{E}_{\mathcal{M}}$. Given a state S and a set of of events E, $\mathrm{GR}(S, E)$ includes all of the events which must be generated by the occurrence of events E in state S and is defined as follows:

$$
\begin{aligned}
\mathrm{GR}(S,E) = \{ & e \in \mathcal{E} \mid e \in E && \textbf{or} \\
& \exists\, e' \in E \text{ s.t. } j = \rho(e'), X \in \mathcal{X}_j, e \in G_j(X, e') \cdot S_{\rho(e)} \models \mathrm{pow}(j, e) \wedge S \models X && \textbf{or} \\
& \exists\, e' \in E, X \in \mathcal{X}_{\rho(e)}, e \in G_{\rho(e)}(X, e') \cdot e \in \mathcal{E}^{\rho(e)}_{viol} \wedge S \models X && \textbf{or} \\
& \exists\, e' \in E \cdot e = viol(e'), S \models \neg\, \mathrm{perm}(e') && \textbf{or} \\
& \exists\, e' \in \mathcal{E}_{\rho(e)}, d \in E \cdot S \models \mathrm{obl}(e', d, e)\}
\end{aligned}
$$

The first condition ensures that events remain generated. The second is responsible for generating those events that are both prescribed by the institutions' event generator and empowered. The third condition deals with violations specified by the event generator, while the fourth generates violations as consequences of events that were not permitted. The final conditions deals with obligations that are not met.

It is easy to see that $\mathrm{GR}(S, E)$ is a monotonic function. This implies that for any given state and a set of events, we can obtain a fixpoint $\mathrm{GR}^{\omega}(S, E)$. In our multi-institutional model, generated events arise from the performance of one *observable event* $e_{obs} \in \mathcal{E}^{\mathcal{M}}_{obs}$ in a given state S. So, to obtain all events that originate from this one event in this state, we simply need $\mathrm{GR}^{\omega}(S, \{e_{obs}\})$.

Event Effects Each fluent is either valid or not in each state of the institution it belongs to. The status of these fluents changes over time according to which generated events have occurred in the previous transition. Events can have two sorts of effects regarding fluents: fluents can be initiated (they become true in the next state) or they can be terminated (they cease to be true in the next state). The combination of all effects generated in a state defines the state transition. The state transition function captures inertia, so all fluents that are not affected in the current state remain valid in the next state.

As mentioned above, given an observable event e_{obs} all events that could have an effect on the state S, are obtained by $\mathrm{GR}^{\omega}(S, \{e_{obs}\})$.

A fluent can be initiated either by any event in the same institution, or by any event in another institution which initiates the fluent, if that institution has the power to initiate the fluent.

$$
\begin{aligned}
\mathrm{INIT}(S, e_{obs}) = \{\ & p \in \mathcal{F}_{\mathcal{M}} \mid \exists\, e \in \mathrm{GR}^{\omega}(S, \{e_{obs}\}), i = \rho(e), X \in \mathcal{X}_i \cdot \\
& p \in \mathcal{F}_i, p \in \mathcal{C}^{\uparrow}_i(X, e), S \models X \textbf{ or} \\
& p \in \mathcal{F}_j, \mathcal{I}_j \neq \mathcal{I}_j, p \in \mathcal{C}^{\uparrow}_i(X, e), S \models X, S \models \mathrm{inipow}(i, p)\}
\end{aligned}
$$

A fluent can be terminated either by an event in the same institution, or by an event in another institution given permission, or if it is a fulfilled obligation in any institution of the multi-institution.

$$\begin{aligned}\mathrm{TERM}(S, e_{obs}) = \{p \in \mathcal{F}_{\mathcal{M}} \mid \exists\, e \in \mathrm{GR}^{\omega}(S, \{e_{obs}\}), i = \rho(e), X \in \mathcal{X}_i \cdot \\ p \in \mathcal{F}_i, p \in \mathcal{C}^{\downarrow}(i, X)e, S \models X \quad \textbf{or} \\ p \in \mathcal{F}_j, \mathcal{I}_j \neq \mathcal{I}_j, p \in \mathcal{C}_i^{\downarrow}(X, e), S \models X, S \models \mathrm{termpow}(i, p) \ \textbf{or} \\ p = \mathrm{obl}(e, d, v) \in \mathcal{F}_{\mathcal{M}} \wedge p \in S \quad \textbf{or} \\ p = \mathrm{obl}(e', e, v) \in \mathcal{F}_{\mathcal{M}} \wedge p \in S\}\end{aligned}$$

Now that we know which fluents need adding or deleting we can define the transition function $TR : \Sigma_{\mathcal{M}} \times \mathcal{E}_{obs}^{\mathcal{M}} \rightarrow \Sigma_{\mathcal{M}}$ as $\mathrm{TR}(\{S_1, \ldots, S_n\}, e_{obs}) = \{S'_1, \ldots, S'_n\}$ such that $S'_i = \{p \in \mathcal{F}_i \mid p \in S, p \notin \mathrm{TERM}(S, e_{obs})$ **or** $p \in \mathrm{INIT}(S, e_{obs})\}$

The first condition models inertia: all fluents which are asserted in the current state persist into the next state, unless they are terminated. The second condition includes fluents which are initiated in the current state.

Ordered Traces Now that we have defined how states may be generated from a previous state and a single observable event, we may define traces and their state evaluations:

- An *ordered trace* is defined as a sequence of observable events $\langle e_0, e_1, \ldots, e_n \rangle$ with $e_i \in \mathcal{E}_{obs}^{\mathcal{M}}, 0 \leq i \leq n$
- The *evaluation of an ordered trace* for a given starting state S_0 is a sequence $\langle S_0, S_1, \ldots S_{n+1} \rangle$ such that $S_{i+1} = \mathrm{TR}(S_i, e_i)$
- Ordered traces and their evaluations allow us to monitor or investigate the evolution of an institution over time. They provide us with the data necessary to answer most queries one might have about a certain (multi-)institution.

A Simple Example: Borrowing

This institution (formalised in Fig. 1) describes when agents may borrow money, when they must pay it back and when they are permitted to leave the interaction. The norm described by the protocol is that when money is borrowed it must be paid back before the agent leaves. Note that the observable events in this institution are not generated by the agents but by the environment in which this institution operates. Also note that the agents will only receive empowerment to leave the institution as soon as they borrow from the institution (line 8). This is to indicate that is useless to leave an agreement before you even started.

The state transition diagram for an instance (with a single agent) of this contract is displayed in Fig. 2. Fluents that are true are included in each state and transitions are labelled with events (generated events are shown in square brackets).

3 InstAL: An Action Language for Describing Institutions

In this section we outline the syntax and semantics of our institutional action language InstAL.

3.1 Syntax

The syntax of our action language consists of a set of declarations which define the types, fluents and events which are supported by the institution and a set of rules which define the operational semantics of the institution. These are summarised by way of the borrowing institution described above.

Given set of agents $Agents$,and multi-institution $\mathcal{M}$ s.t. $a \in Agents$,$\mathcal{I}_{bor}, \mathcal{I}_i \in \mathcal{M}$:

$$\mathcal{E}^{bor}_{obs} = \{\texttt{msg\_borrow}(a), \texttt{msg\_payback}(a), \texttt{msg\_leave}(a)\} \quad (1)$$

$$\mathcal{E}^{bor}_{instact} = \{\texttt{borrow}(a), \texttt{payback}(a), \texttt{leave}(a)\} \quad (2)$$

$$\mathcal{E}^{bor}_{viol} = \{\text{viol}(\texttt{e}) \mid e \in \mathcal{E}^{b}_{obs} \cup \mathcal{E}^{bor}_{instact}\} \cup \{nonpay(a)\} \quad (3)$$

$$\mathcal{D}_{bor} = \{\texttt{loan}(a)\} \quad (4)$$

$$\mathcal{W}_{bor} = \{\text{pow}(i, e) \mid e \in \mathcal{E}^{bor}_{instact}\} \quad (5)$$

$$\mathcal{P}_{bor} = \{\text{perm}(e) \mid e \in \mathcal{E}^{bor}_{obs} \cup \mathcal{E}^{bor}_{instact}\} \quad (6)$$

$$\mathcal{O}_{bor} = \{\text{obl}(e, d, v) \mid e, d \in \mathcal{E}^{bor}_{obs} \cup \mathcal{E}^{bor}_{instact}, v \in \mathcal{E}^{bor}_{inst}\} \quad (7)$$

$$\mathcal{C}^{\uparrow}_{bor}(\mathcal{X}, \mathcal{E}_{\mathcal{M}}): \quad \langle\{\}, \texttt{borrow}(a)\rangle \mapsto \quad \{\text{obl}(\texttt{payback}(a), \texttt{leave}(a), \texttt{nonpay}(a)), \text{pow}(bor, \texttt{payback}(a)), \text{pow}(bor, \texttt{leave}(a)), \texttt{loan}(a)\} \quad (8)$$

$$\mathcal{C}^{\downarrow}_{bor}(\mathcal{X}, \mathcal{E}_{\mathcal{M}}): \quad \langle\{\}, \texttt{borrow}(a)\rangle \mapsto \quad \{\text{pow}(bor, \texttt{borrow}(a))\} \quad (9)$$

$$\langle\{\}, \texttt{payback}(a)\rangle \mapsto \quad \{\text{pow}(bor, \texttt{payback}(a)), \text{pow}(bor, \texttt{leave}(a)), \texttt{loan}(a)\} \quad (10)$$

$$\langle\{\}, \texttt{leave}(a), \rangle \mapsto \quad \{\text{pow}(bor, \texttt{payback}(a)), \text{pow}(bor, \texttt{leave}(a))\} \quad (11)$$

$$\mathcal{G}_{bor}(\mathcal{X}, \mathcal{E}_{\mathcal{M}}): \quad \langle\{\}, \texttt{msg\_borrow}(a)\rangle \mapsto \{\texttt{borrow}(a)\} \quad (12)$$

$$\langle\{\}, \texttt{msg\_payback}(a)\rangle \mapsto \{\texttt{payback}(a)\} \quad (13)$$

$$\langle\{\}, \texttt{msg\_leave}(a)\rangle \mapsto \{\texttt{leave}(a)\} \quad (14)$$

$$S^{bor}_0 = \{\texttt{perm}(\texttt{msg\_borrow}(a)), \texttt{perm}(\texttt{msg\_payback}(a)), \texttt{perm}(\texttt{msg\_leave}(a)), \texttt{perm}(\texttt{borrow}(a)), \texttt{perm}(\texttt{payback}(a)), \texttt{perm}(\texttt{leave}(a)), \texttt{pow}(\texttt{borrow}(a))\} \quad (15)$$

Fig. 1. The formal model of the borrowing scenario

Types Each InstAL specifi cation may contain zero or more types. Types describe a set of atoms which may be applied to the parameters of fluents and events in rule descriptions. The language defi nes four internal types which are grounded automatically by the contents of the specifi cation: (i) The set of events: `Event` (ii) The set of institutions: `Inst` (iii) The set of all fluents `Fluent` (iv) The set of all domain fluents `DFluent` In the example we defi ne one type `Agent` which ranges over the possible subjects of the contract;

```
type  Agent;
```

Event Declarations Each specifi cation may defi ne zero or more event signatures, each of which describes the event's status (observable, action or violation), its (unique) name and the types of any parameters associated with the event.

We defi ne three observable events (Fig. 1, 1) which denote messages associated with: a request to borrow money by an agent (`msg_borrow(..)`), a message describing that the money has been payed back (`msg_payback(...)`) and a message indicating an agent has left the situation (`msg_leave(...)`).

```
observable event msg_borrow(Agent);
observable event msg_payback(Agent);
observable event msg_leave(Agent);
```

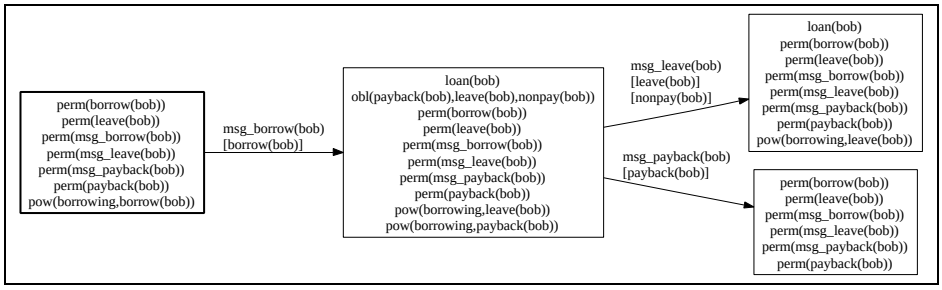

Fig. 2. Reachable states of Borrowing institution for a given agent 'bob'

We define four institutional events (Fig. 1, 2) which denote the effective achievement of borrowing and paying back money and leaving the contract. Additionally we define a violation event (Fig. 1, 3) `nonpay(...)` which is associated with an agent failing to repay borrowed money.

```
action event borrow(Agent);
action event payback(Agent);
action event leave(Agent);
violation event nonpay(Agent);
```

Fluents Fluent declarations define institutional properties which may change over time. A fluent declaration consists of a fluent name, and zero or more fluent parameters, the types of which must be specified.

For instance the following declaration:`fluent owns(Agent,Object);` defines a fluent with name `owns` with two parameters which range over the types `Agent` and `Object` respectively.

In addition to fluents declared in a specification the following types of normative fluents are implicitly defined:

(i) `pow(Inst,Event)`: A given institution is empowered to generated a given event (if no institution is specified then the institution in which the fluent is referenced is assumed).
(ii) `initpow(Inst,DFluent)`,`termpow(Inst,DFluent)`: A given (external) institution has the power to initiate or terminate a given fluent.
(iii) `perm(Event)`: A given event is permitted.
(iv) `obl(Event,Event,Event)` : A given obligation exists.

In the example we define a single institutional domain fluent (Fig. 1, 4) which represents the existence of a loan with respect to some agent.

```
fluent loan(Agent);
```

Rules Each specification may contain zero or more rules, three types of which are available: (i) *Causal rules* which describe when fluents change in response to the occurrence of events. (ii) *Generation rules* which describe when events may be generated. (iii) *Initial rules* which describe the initial state of the institution.

A causal rule consists of (i) *a trigger event* which denotes the event which (may) activate the rule. (ii) *an operation* which indicates whether the rule initiates or terminates the fluents in the rule body. (iii) *a set of fluents* which are initiated or terminated by

the rule. (iv) *a (possibly empty) condition* consisting of an expression describing fluents which must be true in order for the rule to have an effect.

In our example we define the effects of the successful occurrence of the `borrow(..)` event (Fig. 1, 8) as the termination of the power to perform further `borrow` events and creation of the power to pay back and leave the contract, and also the creation of an obligation for the agent to repay the debt before they leave the contract (lest they cause a `nonpay(...)` violation). Borrowing also initiates a loan for the borrowing agent.

```
borrow(A) initiates pow(payback(A)),pow(leave(A)),loan(A),
    obl(payback(A),leave(A),nonpay(A));
```

We similarly define that `borrow(...)` terminates the further power to borrow (Fig. 1, 9) (so borrowing may not occur while a loan exists). Both `payback(...)` and `leave(...)` terminate the power to both payback and leave (Fig. 1, 10-11).

```
borrow(A) terminates pow(borrow(A));
payback(A) terminates pow(payback(A)),pow(leave(A)),loan(A);
leave(A) terminates pow(payback(A)),pow(leave(A));
```

We define three generation rules which associate the performance of the three observable messages with the generation of the corresponding institutional events (Fig. 1, 12-14).

```
msg_borrow(A) generates borrow(A);
msg_payback(A) generates payback(A);
msg_leave(A) generates leave(A);
```

Finally we define the initial state (Fig. 1, 15), in this state all events are permitted, and borrowing is initially empowered.

```
initially perm(msg_borrow(A)),perm(msg_payback(A)),
      perm(msg_leave(A)),perm(borrow(A)),perm(payback(A)),
      perm(leave(A)),pow(borrow(A));
```

Static Properties In addition to fluents which change over time, it is sometimes useful to refer to external properties which will not change during the execution of an institution, but are not known at the time of specification. As with fluents, static properties consist of a name and zero or more typed parameters, for instance the declaration: `static participant(Agent,Inst)` defines a static property with name `participant` and parameters which range over the types `Agent` and `Inst`. This static property indicates which agents are allowed to participate in the institution. It does not imply that the agents cannot come and go at run-time.

Variables Variables are indicated in the language by capitalised strings and may appear in the parameters of fluents and events within rules or within expressions in the conditions of rules (such as `X!=Y`) . Variables are locally scoped to each rule and each variable has a corresponding type, which is computed based on where (in the parameters of a given fluent or event reference) it occurs in the rule.

During processing, a rule containing variables is expanded into a set of rules containing all valid possible assignments of each variable. For example the rule: `sell(X,Y) terminates owns(X,Y);` would be implicitly expanded to variable free rules containing assignments for X and Y based on the parameter types of event `sell` and fluent `owns`. In the case where the condition of a rule rule contains variable expressions (i.e. `A=bob`) then only those variable expansions which satisfy the expressions are generated.

3.2 Model Evaluation

We evaluate properties of our models by performing a transformation of one or more InstAL specifications into answer set programs (see [3] for an extensive overview of answer set programming or [9] for an brief desciption). While the details of this transformation are omitted from this paper, we summarise the process here.

Models are evaluated by taking one or more (related) institutional specifications, a domain description which includes elements of the sets defined in `type` declarations and `static` declarations, a query (see below) and a maximum time interval. This information is then compiled into an answer set program of the form described in [4]. This program may then be solved using an answer set solver such as Smodels[2], yielding zero or more answer sets, each of which represents an ordered trace (up to the maximum time interval) of the institution which matches the query. These traces are then parsed and can be visualised individually or combined into state transition diagrams of the form seen in Figs. 2 and 5.

This process is sound and complete: all requested traces are found as answer sets and all answer sets are valid traces fulfilling the specified conditions. In general the computional complexity of answer set programming is σ_1^P (See [3] for more details). However, the specific chararistics of our program reduces this significantly.

Although ASP can theoretically cope with infinite time, its implementations cannot as the program needs to ground its variables in advance. This implies that we need to specify in advance the number of time steps about which one wishes to reason.

Two mechanisms for specifying queries on the properties of models may be used: (i) for simple queries such as "does it hold that this property is never (or ever) true in the model?" and "what is the state after the performance of this sequence of observable events?" we include a simple query language. In the first case `ever loan(a);` for example would produce all answer sets where a loan is created. (ii) In the second case: `after msg_borrow(a),fineAgent(a),msg_leave(a);` would yield a single answer set describing the series of states (including domain fluents, institution fluents and generated events) brought about by the occurrence of the specified actions. While this query language is useful for simple queries, it represents only a small subset of the possible queries which may be computed using our model. In light of this, queries may also be expressed directly as answer set program rules, for instance:

```
condition:- holdsat(loan(bob),I),not holdsat(loan(bob),J),
   before(I,J),instant(I),instant(J).
compute all { not condition }.
```

would return all traces where a loan was created at some time instant but never settled at some point in the future. In the simple example with a single agent `bob` above this query yields a single trace of length 2: `msg_borrow(bob),msg_leave(bob)` and associated fluents.

3.3 An Extended Example: Contract Enforcement

In the previous section we discussed a single specification of a simple institution for governing loans, however as is clear from Fig. 2 once an agent has violated a loan

[2] `http://www.tcs.hut.fi/Software/smodels/`

Given set of agents $Agents$, contract participants $Part_i \subseteq Agents$ and institutions $\mathcal{M}$ s.t $a_i \in Part_i, enf, i \in \mathcal{M}$:

$$\mathcal{E}_{obs} = \{\texttt{fineAgent}(a_i)\} \tag{16}$$

$$\mathcal{E}_{instact} = \{\texttt{submitContract}(i), \texttt{acceptContract}(i), \texttt{dissolveContract}(i), \texttt{contractViolation}(a_i, i), \texttt{applySanction}(a_i)\} \tag{17}$$

$$\mathcal{E}_{viol} = \{\texttt{badGov}\} \cup \{\text{viol}(\texttt{e}), e \in \mathcal{E}^{enf}_{obs} \cup \mathcal{E}^{enf}_{instact}\} \tag{18}$$

$$\mathcal{D} = \{\texttt{validContract}(i)\} \tag{19}$$

$$\mathcal{W}^{enf} = \{\text{pow}(i, e), e \in \mathcal{E}^{enf}_{instact}\} \tag{20}$$

$$\mathcal{P}^{enf} = \{\text{perm}(e), e \in \mathcal{E}^{enf}_{obs} \cup \mathcal{E}^{enf}_{instact}\} \tag{21}$$

$$\mathcal{O}^{enf} = \{\text{obl}(e, d, v), e, d \in \mathcal{E}^{enf}_{obs} \cup \mathcal{E}^{enf}_{instact}, v \in \mathcal{E}^{enf}_{inst}\} \tag{22}$$

$\mathcal{C}^{\uparrow}_{enf}(\mathcal{X}, \mathcal{E}_{\mathcal{M}})$:

$$\langle\{\}, \texttt{acceptContract}(i)\rangle \mapsto \{\texttt{validContract}(i), \text{pow}(i, \texttt{dissolveContract}(i)), \text{pow}(i, \texttt{contractViolation}(a_i, i))\} \tag{23}$$

$$\langle\{\texttt{validContract}(i)\}, \texttt{contractViolation}(a_i, i)\rangle \mapsto \{\text{pow}(enf, \texttt{applySanction}(a_i)), \text{perm}(\texttt{fineAgent}(a_i)) \text{obl}(\texttt{applySanction}(a_i), \texttt{dissolveContract}(i), \texttt{badGov})\} \tag{24}$$

$\mathcal{C}^{\downarrow}_{enf}(\mathcal{X}, \mathcal{E}_{\mathcal{M}})$:

$$\langle\{\}, \texttt{dissolveContract}(i)\rangle \mapsto \{\texttt{validContract}(i), \text{pow}(i, \texttt{dissolveContract}(i)), \text{pow}(i, \texttt{contractViolation}(a_i, i))\}, \text{perm}(\texttt{fineAgent}(a_i)) \text{pow}(enf, \texttt{applySanction}(a_i)) \tag{25}$$

$$\langle\{\}, \texttt{applySanction}(a_i)\rangle \mapsto \{\text{perm}(\texttt{fineAgent}(a_i)) \text{pow}(enf, \texttt{applySanction}(a_i)) \tag{26}$$

$\mathcal{G}_{enf}(\mathcal{X}, \mathcal{E}_{\mathcal{M}})$:

$$\langle\{\neg\texttt{validContract}(i)\}, \texttt{submitContract}(i)\rangle \mapsto \{\texttt{acceptContract}(i)\} \tag{27}$$

$$\langle\{\}, \texttt{fineAgent}(a_i)\rangle \mapsto \{\texttt{applySanction}(a_i)\} \tag{28}$$

$$S^{enf}_0 = \{\text{pow}(enf, \texttt{acceptContract}(i)), \text{pow}(i, \texttt{submitContract}(i)), \text{perm}(\texttt{submitContract}(i)), \text{perm}(\texttt{acceptContract}(i)), \text{perm}(\texttt{dissolveContract}(i)), \text{perm}(\texttt{applySanction}(a_i))\} \tag{29}$$

Fig. 3. The enforcement institution $enf \in \mathcal{M}$

agreement by leaving before paying, no further action may be taken. In many cases in the real world such violations are delegated to a "higher power" which would impose a sanction. In the following example we demonstrate such an institution which provides a mechanism for enforcing the violation, not only in the borrowing example but also in a broad class of institutions where enforcement is required. [3]

[3] We omit the action language description of this example for space reasons. See `http://www.cs.bath.ac.uk/~occ/instal/` for the source of both examples

Given a set of agents $Agents$ and institutions $\mathcal{M}$ s.t. $a \in Agents$ and $bor \in \mathcal{M}$:

$$\mathcal{C}^{\uparrow}_{bor}(\mathcal{X}, \mathcal{E}_{\mathcal{M}})' : \langle\{\}, \texttt{acceptContract}(bor)\rangle \mapsto \{\mathrm{pow}(e, \texttt{payback}(a))\} \quad (30)$$

$$\mathcal{C}^{\downarrow}_{bor}(\mathcal{X}, \mathcal{E}_{\mathcal{M}})' : \langle\{\}, \texttt{dissolveContract}(bor)\rangle \mapsto \{\mathrm{pow}(e, \texttt{payback}(a))\} \quad (31)$$

$$\mathcal{G}_{bor}(\mathcal{X}, \mathcal{E}_{\mathcal{M}})' : \langle\{\}, \texttt{nopay}(a), \rangle \mapsto \{\texttt{contractViolation}(bor, a)\} \quad (32)$$

$$\langle\{\}, \texttt{borrow}(a)\rangle \mapsto \{\texttt{submitContract}(bor)\} \quad (33)$$

$$\langle\{\}, \texttt{payback}(a)\rangle \mapsto \{\texttt{dissolveContract}(bor)\} \quad (34)$$

$$\langle\{\}, \texttt{fineAgent}(a)\rangle \mapsto \{\texttt{payback}(a)\} \quad (35)$$

Fig. 4. Extensions of the borrowing scenario for contract enforcement

The enforcement institution (enf) (formally described in Fig. 3) describes a single static property `participant(Agent,Inst)` ($Part$ in Fig. 3) which defines when agents are participating in a contract, one domain fluent: `validContract(Inst)` (19) describes when a given institution is considered to be a valid (enforceable) contract. Six event types (17-18) are defined: (i) `submitContract(Inst)` must be generated by the institution in which sanctions are to be enforced. It generates an `acceptContract(..)` event if the submitted institution is not already considered a valid contract (27), and initiates `validContract(..)` for the accepted contract and the power for that contract to generate contract violations in the enforcement institution. (ii) `acceptContract(Inst)` describes when a contract is is treated as valid, initiating (23) the `validContract(i)` fluent also the power for the new contract to dissolve itself and generate contract violation events in the enforcement institution. (iii) `contractViolation(Agent)` must be generated by the contract and when generated, initiates an obligation in the enforcement institution to apply a sanction to that agent before the contract is terminated, or be subject to a `badViol` violation. (iv) `fineAgent(Agent)` is an observable event which stands for the imposition of a fine. In the case that a valid contract exists with this agent this event generates an `applySanction(..)` event (28) in the institution (if this event is empowered). (v) `dissolveContract(Inst)` terminates all powers granted to the contract by its acceptance and also terminates the validity of the contract (25).

In order to make the borrowing contract enforceable by this institution we extend it as follows (Formalised in Fig. 4): borrowing money creates a contract in the enforcement institution (33), paying back money dissolves the contract (34), not paying back the money generates a contract violation (32) and fining an agent in the enforcement institution pays back the debt (35). Additionally the creation and dissolution of a contract in the enforcement institution initiate and terminate the power for that institution to pay back debts (30-31) finally the initial state is extended to permit and empower the generation of re-payment (not shown).

Fig. 5 shows the reachable states for a combined model (with one borrower) of the enforcement and borrowing institutions. The ASP query in Section 3.2 above, yields no answer sets, indicating that there are no traces in the generated model (up to the search length) where a loan is not repaid.

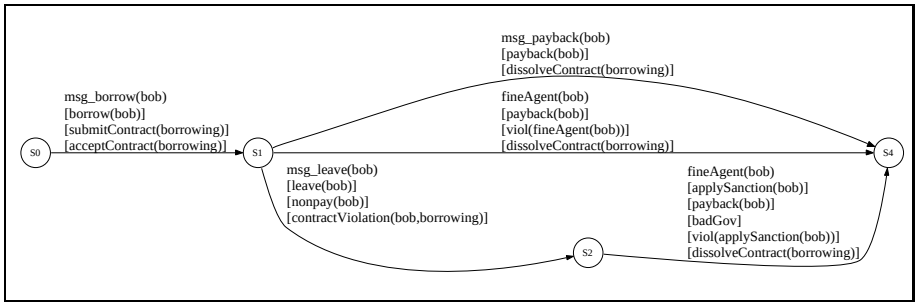

Fig. 5. States of the combined borrowing and enforcement institution

4 Discussion and Related Work

The use of common-sense reasoning tools such as action languages and answer set programing for reasoning about normative systems and agent-based systems in general has been studied extensively in the literature and a complete analysis is beyond the scope of this paper, however a number of recent studies merit discussion.

The Event Calculus (EC) [13, 14] is a declarative logic that reinterprets the Situation Calculus to capture when and how states change in response to external events. EC has been used to model both the behaviour of commitments [20] among agents in order to build interaction protocols, corresponding to the regulatory aspects of the work described above, as well as more general social models such as those described in [12]. From a technical point of view, our approach essentially has a kind of duality compared to EC, in that the basis for the model is events rather than states. In itself, this offers no technical advantage although we believe that being able to express violations in terms of events rather than states better captures their nature. More signifi cant are the consequences of the grounding in ASP: (i) For the most part the state and event models are equivalent with respect to properties such as induction and abduction, but non-monotonicity is inherent in ASP and so resort to the tricky process of circumscription is avoided. (ii) Likewise, reasoning about defaults requires no special treatment in ASP. (iii) The consequence rules of our specifi cation have equivalents in EC, but the event generation rules do not. (iv) The state of a fluent is determined by its truth-value in the ASP interpretation, whereas EC (typically) has to encode this explicitly using two predicates. (v) Inertia in EC is axiomatic, whereas in our approach it follows from the application of the TR operator—although there is a strong syntactic similarity (perhaps compounded by using the same terminology!) the philosophy is different. (vi) ASP allows a wider variety of queries than is typically provided in EC implementations but space constraints do not allow the full illustration of this aspect here. We also note that, EC is much more general, in that it is aimed at capturing arbitrary narrative, while the InstAL language we have presented is more like a domain-specifi c language that allows only the expression of institutional issues and in that sense is more restrictive than EC.

[8] use EC to represent the specifi cation of contracts. Their approach of dealing with contract is similar to ours but with some important differences with respect to the broader picture of multi-institutions: (i) any formalisation of pow/permissions/obligation is ommitted from their specifi cation and left as domain dependent concepts which are modelled using XML (ii) this means that that their approach does not have conven-

tional generation of events/obligations/permissions explicitly, only their effects (iii) in their work the authors are just concerned with monitoring the state, not investigating other properties (i.e. planning/verifi caition), although these may also be possible

Artikis et al. in [1, 2, 12] describe a system for the specifi cation of normative social systems in terms of power, empowerment and obligation. This is formalized using both the event calculus [13] and a subset of the action language $C+$ [6]. The notions of power and empowerment are equivalent in both systems, but additionally we introduces violation as events and our modelling of obligations differs in that (i) they are deadline-sensitive, and (ii) can raise a violation if they are not met in time. Violations greatly improve the capacity to model institutions, but it should be remembered that institutional modelling was (apparently) not Artikis's goal. Likewise, although the interpretation of $C+$ using the CCalc tool gives rise to similar reasoning capabilities (with similar complexity) to ASP, we believe our approach, including violations, provides a more intuitive and natural way of expressing social constraints involving temporal aspects. A further advantage is in the formulation of queries, where ASP makes it possible to encode queries similar to those found in (bounded) temporal logic model checking, whereas, as noted above, queries on action languages are constrained by the action language implementation. The other notable difference is once again, our focus on events rather than states, which we have discussed at some length above.

The syntax and underlying semantics of the action language we present here are similar to those of the $\mathcal{C}+$ language in [6]. Besides internal support for the semantics of institutions our approach specifi es the effects of actions (in particular the termination of inertial fluents) in a different way. For example, in $\mathcal{C}+$ the rules "`a causes f`.`a causes` $\neg$`f`" will necessarily lead to `a` being non-executable, the corresponding statements `a initiates f; a terminates f` in our language do not effect the ability of `a` to occur or be generated and can be handled consistently (leading to `f` holding immediately after `a`); this is similar to the treatment of fluents in the event calculus [13]. The choice of our semantics stems from a desire to assimilate actions in the real world rather than model them accurately, in which case, the institution should be able to generate consistently a consequent institutional state (albeit one in which no effect has occurred), regardless of the originating event.

While their work does not consider multiple institutions, $\mathcal{C}+$ could be used for this purpose. This would, however come at the cost of the semantic and syntactic checking of the institutional extensions which we provide. $\mathcal{C}+$ does offer some syntactic extensions, which would lead more concise specifi cations. Of particular interest are *multivalued fluents* (where a fluent is multi-valued rather than boolean) and *Dynamic laws* e.g. "`caused f1 if f2 `" which allow for the state of fluents to be expressed indirectly as a function of other (inertial) fluents. The integration of either of these features into InstAL appears straightforward and is left for future work.

References

[1] A. Artikis, M. Sergot, and J. Pitt. An executable specification of an argumentation protocol. In *Proceedings of conference on artificial intelligence and law (icail)*, pages 1–11. ACM Press, 2003.

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

[3] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[4] O. Cliffe, M. De Vos, and J. Padget. Answer set programming for representing and reasoning about virtual institutions. In *Computational Logic in Multi-Agent Systems (CLIMA-VII)*, May 2006. To appear.

[5] O. Cliffe, M. De Vos, and J. Padget. Specifying and analysing agent-based social institutions using answer set programming. In *Selected revised papers from the workshops on Agent, Norms and Institutions for Regulated Multi-Agent Systems (ANIREM) and Organizations and Organization Oriented Programming (OOOP) at AAMAS'05*, volume 3913 of *LNCS*. Springer Verlag, 2006. To appear.

[6] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence, Vol. 153, pp. 49-104*, 2004.

[7] M. Esteva, J. Padget, and C. Sierra. Formalizing a language for institutions and norms. In M. Tambe and J.-J. Meyer, editors, *Intelligent Agents VIII*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2001.

[8] A. D. H. Farrell, M. J. Sergot, M. Sallé, and C. Bartolini. Using the event calculus for tracking the normative state of contracts. *International Journal of Cooperative Information Systems*, 14(2 & 3):99–129, 2005.

[9] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of fifth logic programming symposium*, pages 1070–1080. MIT PRESS, 1988.

[10] M. Gelfond and V. Lifschitz. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998.

[11] John R. Searle. *The Construction of Social Reality*. Allen Lane, The Penguin Press, 1995.

[12] L. Kamara, A. Artikis, B. Neville, and J. Pitt. Simulating computational societies. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Proceedings of workshop on engineering societies in the agents world (esaw)*, LNCS 2577, pages 53–67. Springer, 2003.

[13] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986.

[14] R. A. Kowalski and F. Sadri. Reconciling the event calculus with the situation calculus. *Journal of Logic Programming*, 31(1–3):39–58, Apr.–June 1997.

[15] Pablo Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona, 1997.

[16] J. Padget and R. Bradford. A π-calculus model of the spanish fishmarket. In *Proceedings of AMET'98*, volume 1571 of *Lecture Notes in Artificial Intelligence*, pages 166–188. Springer Verlag, 1999.

[17] J.-A. Rodríguez, P. Noriega, C. Sierra, and J. Padget. FM96.5 A Java-based Electronic Auction House. In *Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97)*, pages 207–224, London, UK, Apr. 1997. ISBN 0-9525554-6-8.

[18] J. A. Rodriguez-Aguilar. *On the Design and Construction of Agent-mediated Institutions*. PhD thesis, Universitat Autonoma de Barcelona, 2001.

[19] J. V. Salceda. *The role of Norms and Electronic Institutions in Multi-Agent Systems applied to complex domains*. PhD thesis, Technical University of Catalonia, 2003.

[20] P. Yolum and M. P. Singh. Flexible protocol specification and execution: applying event calculus planning using commitments. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 527–534. ACM Press, 2002.