Logic-Based Electronic Institutions

Wamberto W. Vasconcelos

Department of Computing Science, University of Aberdeen
Aberdeen AB24 3UE, United Kingdom
wvasconcelos@acm.org

Abstract. We propose a logic-based rendition of electronic institutions – these are means to specify open agent organisations. We employ a simple notation based on first-order logic and set theory to represent an expressive class of electronic institutions. We also provide a formal semantics to our constructs and present a distributed implementation of a platform to enact electronic institutions specified in our formalism.

1 Introduction

In this paper we propose a logical formalism that allows the representation of a useful class of protocols involving many agents. This formalism combines first-order logic and set theory to allow the specification of interactions among agents, whether an auction, a more sophisticated negotiation or an argumentation framework. We introduce and exploit the logic-based formalism within the context of electronic institutions: these are means to modularly describe open agent organisations [6]. As well as providing a flexible syntax for interactions, we also formalise their semantics via the construction of models.

Current efforts at standardising agent communication languages like KIF and KQML [12] and FIPA-ACL [7] do not cater for dialogues: they do not offer means to represent relationships among messages. Work on dialogues (*e.g.* [11], [15] and [21]), on the other hand, prescribe the actual format, meaning and ultimate goal of the interactions. Our effort aims at providing engineers with a notation for specifying interactions among the components of a Multi-Agent System (MAS, for short), but which allows *relationships* to be forged among the interactions. A typical interaction we are able to express in our formalism is "all seller agents advertise their goods; after this, all buyer agents send their offers for the goods to the respective seller agent". In this interaction, it is essential that the buyer agents send offers to the appropriate seller agents, that is, each seller agent should receive an appropriate offer to the good(s) it advertised.

This paper is structured as follows. In Section 2 we describe the syntax and semantics of our proposed logic-based formalism to describe protocols. In Section 3 we introduce a definition of electronic institutions using our logic-based notation for protocols giving their formal meaning; in that section we illustrate our approach with a practical example and we describe how we implemented a platform to enact electronic institutions expressed in our formalism. Finally, in Section 4 we draw conclusions and give directions for future work.

2 A Set-Based Logic $\mathcal{L}$ for Protocols

In this section, we describe a set-based first-order logic $\mathcal{L}$ with which we can define protocols. Our proposed logic provides us with a compact notation to formally describe relationships among messages in a protocol. Intuitively, these constructs define (pre- and post-) conditions that should hold as agents follow a protocol.

We aim at a broad class of protocols in which many-to-many interactions (and, in particular, one-to-one, one-to-many and many-to-one) can be formally expressed. The protocols are *global* in the sense that they describe any and all interactions that may take place in the MAS. One example of the kind of interactions we want to be able to express is "an agent x sends a message to another agent y offering an item k for sale; agent y replies to x's message making an offer n to buy k" and so on. We define $\mathcal{L}$ as below:

Definition 1. *$\mathcal{L}$ consists of formulae $Qtf\ (Atfs \Rightarrow SetCtrs)$ where Qtf is the quantification, $Atfs$ is a conjunction of atomic formulae and $SetCtrs$ is a conjunction of set constraints.*

Qtf provides our constructs with universal and existential quantification over (finite) sets; *Atfs* expresses atomic formulae that must hold true and *SetCtrs* represents set constraints that (are made to) hold true. We define the classes of constructs *Qtf*, *Atfs* and *SetCtrs* in the sequel. We refer to a well-formed formulae of $\mathcal{L}$ generically as *Fml*.

We shall adopt some notational conventions in our formulae. Sets will be represented by words starting with capital letters and in this typefont, as in, for example "S", "Set" and "Buyers". Variables will be denoted by words starting with capital letters in *this typefont*, as in, for example, "X", "Var" and "$Buyer$". We shall represent constants by words starting with non-capital letters in *this font*; some examples are "*a*" and "*item*". We shall assume the existence of a recursively enumerable set *Vars* of variables and a recursively enumerable set *Consts* of constants.

In order to define the class *Atfs* of atomic formulae conjunctions, we first put forth the concept of *terms*:

Definition 2. *All elements from Vars and Consts are in Terms. If $t_1, \ldots, t_n$ are in Terms, then $f(t_1, \ldots, t_n)$ is also in Terms, f being a function symbol.*

The class *Terms* is thus defined recursively, based on variables and constants and their combination with functional symbols. An example of a term using our conventions is $enter(buyer)$. We can now define the class *Atfs*:

Definition 3. *If $t_1, \ldots, t_n$ are Terms, then $p(t_1, \ldots, t_n)$ is an atomic formula (or, simply, an atf), where p is any predicate symbol. A special atomic formula is defined via the "=" symbol, as $t_1 = t_2$. The class Atfs consists of all atfs; furthermore, for any Atf_1 and Atf_2 in Atfs, $Atf_1 \wedge Atf_2$ is also in Atfs.*

This is another recursive definition: the basic components are the simple atomic formulae built with terms. These components (and their combinations) can be put together as conjuncts.

We now define the class of *set constraints*. These are restrictions on set operations such as union, intersection, Cartesian product and set difference [8]:

Definition 4. *Set constraints are conjunctions of set operations, defined by the following grammar:*

$$\begin{aligned}
\mathit{SetCtrs} &\rightarrow \mathit{SetCtrs} \wedge \mathit{SetCtrs} \mid (\mathit{SetCtrs}) \mid \mathit{MTest} \mid \mathit{SetProp}\\
\mathit{MTest} &\rightarrow \mathit{Term} \in \mathit{SetOp} \mid \mathit{Term} \notin \mathit{SetOp}\\
\mathit{SetProp} &\rightarrow \mathit{card}(\mathit{SetOp})\ \mathit{Op}\ \mathbb{N} \mid \mathit{card}(\mathit{SetOp})\ \mathit{Op}\ \mathit{card}(\mathit{SetOp}) \mid \mathit{SetOp} = \mathit{SetOp}\\
\mathit{Op} &\rightarrow = \ \mid\ > \ \mid\ \geq \ \mid\ < \ \mid\ \leq\\
\mathit{SetOp} &\rightarrow \mathit{SetOp} \cup \mathit{SetOp} \mid \mathit{SetOp} \cap \mathit{SetOp} \mid \mathit{SetOp} - \mathit{SetOp}\\
&\quad \mid \mathit{SetOp} \times \mathit{SetOp} \mid (\mathit{SetOp}) \mid \mathsf{Set} \mid \emptyset
\end{aligned}$$

MTest is a *membership test*, that is, a test whether an element belongs or not to the result of a set operation *SetOp* (in particular, to a specific set). *SetProp* represents the *set properties*, that is, restrictions on set operations as regards to their size (*card*) or their contents. $\mathbb{N}$ is the set of natural numbers. *Op* stands for the allowed operators of the set properties. *SetOp* stands for the *set operations*, that is, expressions whose final result is a set. An example of a set constraint is $B \in \mathsf{Buyers} \wedge \mathit{card}(\mathsf{Buyers}) \geq 0 \wedge \mathit{card}(\mathsf{Buyers}) \leq 10$. We may, alternatively, employ $|\mathsf{Set}|$ to refer to the cardinality of a set, that is, $|\mathsf{Set}| = \mathit{card}(\mathsf{Set})$. Additionally, in order to simplify our set expressions and improve their presentation, we can use $0 \leq |\mathsf{Buyers}| \leq 10$ instead of the previous expression.

Finally, we define the quantifications *Qtf*:

Definition 5. *The quantification Qtf is defined as:*

$$\begin{aligned}
\mathit{Qtf} &\rightarrow \mathit{Qtf}'\ \mathit{Qtf} \mid \mathit{Qtf}'\\
\mathit{Qtf}' &\rightarrow Q\ \mathit{Var} \in \mathit{SetOp} \mid Q\ \mathit{Var} \in \mathit{SetOp}, \mathit{Var} = \mathit{Term}\\
Q &\rightarrow \forall \mid \exists \mid \exists!
\end{aligned}$$

Where $\mathit{Term} \in \mathit{Terms}$ *and* $\mathit{Var} \in \mathit{Vars}$.

We pose an important additional restriction on our quantifications: either *Var* or subterms of *Term* must occur in $(\mathit{Atfs} \Rightarrow \mathit{SetCtrs})$.

Using the typographic conventions presented above, we can now build correct formulae; an example is $\exists B \in \mathsf{Ags}\,(m(B, \mathit{adm}, \mathit{enter}(\mathit{buyer})) \Rightarrow (B \in \mathsf{Bs} \wedge 1 \leq |\mathsf{Bs}| \leq 10))$. To simplify our formulae, we shall also write quantifications of the form $\mathit{Qtf}\ \mathit{Var} \in \mathit{SetOp}, \mathit{Var} = \mathit{Term}$ simply as $\mathit{Qtf}\ \mathit{Term} \in \mathit{SetOp}$. For instance, $\forall X \in \mathsf{Set}, X = f(a, Z)$ will be written as $\forall f(a, Z) \in \mathsf{Set}$.

2.1 The Semantics of $\mathcal{L}$

In this section we show how *Fml* is mapped to truth values $\top$ (true) or $\bot$ (false). For that, we first define the *interpretation* of our formulae:

Definition 6. *An interpretation* $\Im$ *for Fml is the pair* $\Im = (\sigma, \Omega)$ *where* σ *is a possibly empty set of ground atomic formulae (i.e. atfs without variables) and* Ω *is a set of sets.*

Intuitively our interpretations provide in σ what is required to determine the truth value of $\mathit{Qtf}(\mathit{Atfs})$ and in Ω what is needed in order to assign a truth value to $\mathit{Qtf}(\mathit{SetCtrs})$.

We did not include in our definition of interpretation above the notion of *universe of discourse* (also called *domain*) nor the usual mapping between constants and elements of this universe, neither the mapping between function and predicate symbols of the formula and functions and relations in the universe of discourse [4, 13]. This is because we are only interested in the relationships between *Atfs* and *SetCtrs* and how we can automatically obtain an interpretation for a given formula. However, we can define the union of all sets in Ω as our domain. It is worth mentioning that the use of a set of sets to represent Ω does not cause undesirable paradoxes: since we do not allow the formulae in $\mathcal{L}$ to make references to Ω, but only to sets in Ω, this will not happen.

The semantic mapping $\mathbf{k} : \mathit{Fml} \times \Im \mapsto \{\top, \bot\}$ is:

1. $\mathbf{k}(\forall \mathit{Terms} \in \mathit{SetOp}\ \mathit{Fml}, \Im) = \top$ iff $\mathbf{k}(\mathit{Fml}|_{e}^{\mathit{Terms}}, \Im) = \top$ for all $e \in \mathbf{k}'(\mathit{SetOp}, \Im)$
   $\mathbf{k}(\exists \mathit{Terms} \in \mathit{SetOp}\ \mathit{Fml}, \Im) = \top$ iff $\mathbf{k}(\mathit{Fml}|_{e}^{\mathit{Terms}}, \Im) = \top$ for some $e \in \mathbf{k}'(\mathit{SetOp}, \Im)$
   $\mathbf{k}(\exists! \mathit{Terms} \in \mathit{SetOp}\ \mathit{Fml}, \Im) = \top$ iff $\mathbf{k}(\mathit{Fml}|_{e}^{\mathit{Terms}}, \Im) = \top$ for a single $e \in \mathbf{k}'(\mathit{SetOp}, \Im)$
2. $\mathbf{k}((\mathit{Atfs} \Rightarrow \mathit{SetCtrs}), \Im) = \bot$ iff $\mathbf{k}(\mathit{Atfs}, \Im) = \top$ and $\mathbf{k}(\mathit{SetCtrs}, \Im) = \bot$
3. $\mathbf{k}(\mathit{Atfs}_1 \wedge \mathit{Atfs}_2, \Im) = \top$ iff $\mathbf{k}(\mathit{Atfs}_1, \Im) = \mathbf{k}(\mathit{Atfs}_2, \Im) = \top$
   $\mathbf{k}(\mathit{Atf}, \Im) = \top$ iff $\mathit{Atf} \in \sigma, \Im = (\sigma, \Omega)$
4. $\mathbf{k}(\mathit{SetCtrs}_1 \wedge \mathit{SetCtrs}_2, \Im) = \top$ iff $\mathbf{k}(\mathit{SetCtrs}_1, \Im) = \mathbf{k}(\mathit{SetCtrs}_2, \Im) = \top$
5. $\mathbf{k}(\mathit{Terms} \in \mathit{SetOp}, \Im) = \top$ iff $\mathit{Terms} \in \mathbf{k}'(\mathit{SetOp}, \Im)$;
   $\mathbf{k}(\mathit{Terms} \notin \mathit{SetOp}, \Im) = \top$ iff $\mathit{Terms} \notin \mathbf{k}'(\mathit{SetOp}, \Im)$
   $\mathbf{k}(|\mathit{SetOp}|\ \mathit{Op}\ \mathbb{N}, \Im) = \top$ iff $|\mathbf{k}'(\mathit{SetOp}, \Im)|\ \mathit{Op}\ \mathbb{N}$ holds.
   $\mathbf{k}(|\mathit{SetOp}_1|\ \mathit{Op}\ |\mathit{SetOp}_2|, \Im) = \top$ iff $|\mathbf{k}'(\mathit{SetOp}_1, \Im)|\ \mathit{Op}\ |\mathbf{k}'(\mathit{SetOp}_2, \Im)|$ holds.
   $\mathbf{k}(\mathit{SetOp}_1 = \mathit{SetOp}_2, \Im) = \top$ iff $\mathbf{k}'(\mathit{SetOp}_1, \Im) = \mathbf{k}'(\mathit{SetOp}_2, \Im)$

In item 1 we address the three quantifiers over *Fml* formulae, where $\mathit{Fml}|_{e}^{\mathit{Terms}}$ is the result of replacing every occurrence of *Terms* by e in *Fml*. Item 2 describes the usual meaning of the right implication. Item 3 formalises the meaning of conjunctions *Atfs* and the basic case for individual atomic formulae – these are only considered true if they belong to the associated set σ of the interpretation $\Im$. Item 4 formalises the meaning of the conjunct and disjunct operations over set constraints *SetCtrs* and the basic membership test to the result of a set operation *SetOp*. Item 5 describes the truth-value of the distinct set properties *SetProp*. These definitions describe only one case of the mapping: since ours is a total mapping, the situations which are not described represent a mapping with the remaining value $\top$ or $\bot$.

The auxiliary mapping $\mathbf{k}' : \mathit{SetOp} \times \Im \mapsto \mathsf{Set}$ in $\Omega, \Im = (\sigma, \Omega)$, referred to above and which gives meaning to the set operations is thus defined:

1. $\mathbf{k}'(\mathit{SetOp}_1 \cup \mathit{SetOp}_2, \Im) = \{e \mid e \in \mathbf{k}'(\mathit{SetOp}_1, \Im) \text{ or } e \in \mathbf{k}'(\mathit{SetOp}_2, \Im)\}$
2. $\mathbf{k}'(\mathit{SetOp}_1 \cap \mathit{SetOp}_2, \Im) = \{e \mid e \in \mathbf{k}'(\mathit{SetOp}_1, \Im) \text{ and } e \in \mathbf{k}'(\mathit{SetOp}_2, \Im)\}$
3. $\mathbf{k}'(\mathit{SetOp}_1 - \mathit{SetOp}_2, \Im) = \{e \mid e \in \mathbf{k}'(\mathit{SetOp}_1, \Im) \text{ and } e \notin \mathbf{k}'(\mathit{SetOp}_2, \Im)\}$
4. $\mathbf{k}'(\mathit{SetOp}_1 \times \mathit{SetOp}_2, \Im) = \{(e_1, e_2) \mid e_1 \in \mathbf{k}'(\mathit{SetOp}_1, \Im) \text{ and } e_2 \in \mathbf{k}'(\mathit{SetOp}_2, \Im)\}$
5. $\mathbf{k}'((\mathit{SetOp}), \Im) = (\mathbf{k}'(\mathit{SetOp}, \Im))$.
6. $\mathbf{k}'(\mathsf{Set}, \Im) = \{e \mid e \in \mathsf{Set} \text{ in } \Omega, \Im = (\sigma, \Omega)\}$, $\mathbf{k}'(\emptyset, \Im) = \emptyset$.

The 4 set operations are respectively given their usual definitions [8]. The meaning of a particular set Set is its actual contents, as given by Ω in $\Im$. Lastly, the meaning of an empty set $\emptyset$ in a set operation is, of course, the empty set.

We are interested in *models* for our formulae, that is, interpretations that map *Fml* to the truth value $\top$ (true). We are only interested in those interpretations in which *both* sides of the "$\Rightarrow$" in the *Fml*'s hold true. Formally:

Definition 7. *An interpretation $\Im = (\sigma, \Omega)$ is a model for a formula $\mathit{Fml} = \mathit{Qtf}(\mathit{Atfs} \Rightarrow \mathit{SetCtrs})$, denoted by $\mathbf{m}(\mathit{Fml}, \Im)$ iff σ and Ω are the smallest possible sets such that $\mathbf{k}(\mathit{Qtf}\ \mathit{Atfs}, \Im) = \mathbf{k}(\mathit{Qtf}\ \mathit{SetCtrs}, \Im) = \top$.*

The scenarios arising when the left-hand side of the *Fml* is false do not interest us: we want this formalisation to restrict the meanings of our constructs only to those desirable (correct) ones. The study of the anomalies and implications caused by not respecting the restrictions of a protocol albeit important is not in the scope of this work.

We now define the extension of an interpretation, necessary to build models for more than one formula *Fml*:

Definition 8. *$\Im' = (\sigma', \Omega')$ is an extension of $\Im = (\sigma, \Omega)$ which accommodates Fml, denoted by $\mathbf{ext}(\Im, \mathit{Fml}) = \Im'$, iff $\mathbf{m}(\mathit{Fml}, \Im''), \Im'' = (\sigma'', \Omega'')$ and $\sigma' = \sigma \cup \sigma'', \Omega' = \Omega \cup \Omega''$.*

3 Logic-Based Electronic Institutions

In the same way that social institutions, such as a constitution of a country or the rules of a club, are somehow forged (say, in print or by common knowledge), the laws that should govern the interactions among heterogeneous agents can be defined by means of electronic institutions (e-institutions, for short) [5, 6, 14, 16]. E-institutions are non-deterministic finite-state machines describing possible interactions among agents. The interactions are only by means of message exchanges, that is, messages that are sent and received by agents. E-institutions define communication protocols among agents with a view to achieving global and individual goals.

Although different formulations of e-institutions can be found in the literature [5, 6, 14, 16, 19], they all demand additional informal explanations concerning the precise meaning of its constructs. In an e-institution the interactions among agents are described as finite-state machines with messages labelling the edges between two states. A simple example is graphically depicted in Fig. 1 where two agents x and y engage in a simple two-step conversation

$$\begin{tikzcd} w_1 \arrow[r, "m(x,y,sell(k))"] & w_2 \arrow[r, "m(y,x,offer(k,n))"] & w_3 \end{tikzcd}$$

Fig. 1: Protocol as a Finite-State Machine

– to save space, we have represented messages as atomic formulae of the form $m(\mathit{Sender}, \mathit{Addressee}, \mathit{Conts})$, meaning that *Sender* is sending to *Addressee* message *Conts*; alternative formats such as FIPA-ACL [7] could be used instead. Agent x informs agent y that it wants to sell item k and y replies with an offer n. In the example above we employed variables x, y, k and n but it is not clear what their actual meaning is: is x the same in both edges? is it just *one* agent x or can many agents follow the transition? It is not clear from the notation only what the meaning of the label is. Surely, informal explanations could solve any ambiguity, but by tacitly assuming the meaning of constructs (*i.e.* "hardwiring" the meaning to the syntax), then variations cannot be offered. For instance, if we assume that the variables in Fig. 1 are universally quantified, then it is not possible to express the existential quantification and vice-versa. Similar expressiveness losses occur when other assumptions are made.

We have incorporated our proposed logic $\mathcal{L}$ to the definition of e-institutions. In this combination, constructs of $\mathcal{L}$ label edges of finite-state machines. This allows for precisely defined and expressive edges thus extending the class of e-institutions one can represent. Furthermore, by embedding $\mathcal{L}$ within e-institutions, we can exploit the model-theoretic issues in an operational framework.

3.1 Scenes

Scenes are the basic components of an e-institution, describing interactions among agents:

Definition 9. *A scene is* $\mathbf{S} = \langle R, W, w_0, W_f, \mathit{WA}, \mathit{WE}, f^{\mathit{Guard}}, \mathit{Edges}, f^{\mathit{Label}} \rangle$ *where*
- $R = \{r_1, \ldots, r_n\}$ *is the set of roles;*
- $W = \{w_0, \ldots, w_m\}$ *is a finite, non-empty set of states;*
- $w_0 \in W$ *is the initial state;*
- $W_f \subseteq W$ *is the non-empty set of final states;*
- WA *is a set of sets* $\mathit{WA} = \{\mathit{WA}_r \subseteq W \mid r \in R\}$ *where each* WA_r, $r \in R$, *is the set of access states for role r;*

– *WE is a set of sets $WE = \{WE_r \subseteq W \mid r \in R\}$ where each WE_r, $r \in R$, is the set of exit states for role r;*
– *$f^{Guard} : WA_r \mapsto Fml$ and $f^{Guard} : WE_r \mapsto Fml$ associates with each access state WA_r and exit state WE_r of role r a formula Fml.*
– *$Edges \subseteq W \times W$ is a set of directed edges;*
– *$f^{Label} : Edges \mapsto Fml$ associates each element of $Edges$ with a formula Fml.*

This definition is a variation of that found in [19]. We have added to access and exit states, via function f^{Guard}, explicit restrictions formulated as formulae of $\mathcal{L}$. The labelling function f^{Label} is defined similarly, but mapping *Edges* to our formulae *Fml*.

3.2 Transitions

The scenes, as formalised above, are where the communication among agents actually take place. However, individual scenes can be part of a more complex context in which specific sequences of scenes have to be followed. For example, in some kinds of electronic markets, a scene where agents meet other agents to choose their partners to trade is followed by a scene where the negotiations actually take place. We define *transitions* as a means to connect and relate scenes:

Definition 10. *A transition is $\mathbf{T} = \langle CI, w_a, Fml, w_e, CO\rangle$ where*

– *$CI \subseteq \bigcup_{i=1}^{n}(WE_i \times w_a)$, is the set of connections into the transition, $WE_i, 1 \leq i \leq n$ being the sets of exit states for all roles from all scenes;*
– *w_a is the access state of the transition;*
– *w_e is the exit state of the transition;*
– *Fml, a formula of $\mathcal{L}$, labels the pair $(w_a, w_e) \mapsto Fml$;*
– *$CO \subseteq \bigcup_{j=1}^{m}(w_e \times WA_j)$, is the set of connections out of the transition, $WA_j, 1 \leq j \leq m$ being the sets of access states for all roles onto all scenes.*

A transition has only two states w_a, its access state, and w_e, its exit state, and a set of connections CI relating the exit states of scenes to w_a and a set of connections CO relating w_e to the access states of scenes. The conditions under which agents are allowed to move from w_a to w_e are specified by a formula *Fml* of our set-based logic, introduced above.

Transitions can be seen as simplified scenes where agents' movements can be grouped together and synchronised out of a scene and into another one. The roles of agents may change, as they go through a transition. An important feature of transitions lies in the kinds of formula *Fml* we are allowed to use. Contrary to scenes, where there can only be references to constructs within the scene, within a transition we can make references to constructs of any scene that connects to the transition. This difference is formally represented by the semantics of e-institutions below.

3.3 $\mathcal{L}$-Based E-Institutions

Our e-institutions are collections of scenes and transitions:

Definition 11. *An e-institution is $\mathbf{E} = \langle Scenes, \mathbf{S}_0, \mathbf{S}_f, Trans\rangle$ where*

– *$Scenes = \{\mathbf{S}_0, \ldots, \mathbf{S}_n\}$ is a finite and non-empty set of scenes;*
– *$\mathbf{S}_0 \in Scenes$ is the root scene;*
– *$\mathbf{S}_f \in Scenes$ is the output scene;*
– *$Trans = \{\mathbf{T}_0, \ldots, \mathbf{T}_m\}$ is a finite and non-empty set of transitions;*

We shall impose the restriction that the transitions of an e-institution can only connect scenes from the set *Scenes*, that is, for all $\mathbf{T} \in \mathit{Trans}$, $\mathit{CI} \subseteq \bigcup_{i=0}^{n}(\mathit{WE}_i \times w_a), i \neq f$ (the exit states of the output scene can not be connected to a transition) and $\mathit{CO} \subseteq \bigcup_{j=1}^{n}(w_e \times \mathit{WA}_j)$ (the access state of the root scene cannot be connected to a transition).

For the sake of simplicity, we have not included in our definition above the *normative rules* [5] which capture the obligations agents get bound to as they exchange messages. We are aware that this makes our definition above closer to the notion of *performative structure* [5] rather than an e-institution.

3.4 Models for $\mathcal{L}$-Based E-Institutions

In this section we introduce models for scenes, transitions and e-institutions using the definitions above.

A model for a scene is built using the formulae that label edges connecting the initial state to a final state. The formulae guarding access and exit states are also taken into account: they are used to extend the model of the previous formulae and this extension is further employed with the formula connecting the state onwards. Since there might be more than one final state and more than one possible way of going from the initial state to a final state, models for scenes are not unique. More formally:

Definition 12. *An interpretation $\Im$ is a model for a scene $\mathbf{S} = \langle R, W, w_0, W_f, \mathit{WA}, \mathit{WE}, f^{\mathit{Guard}}, \mathit{Edges}, f^{\mathit{Label}}\rangle$, given an initial interpretation $\Im_0$, denoted by $\mathbf{m}(\mathbf{S}, \Im)$, iff $\Im = \Im_n$, where:*

- *$f^{\mathit{Label}}(w_{i-1}, w_i) = \mathit{Fml}_i, 1 \leq i \leq n, w_n \in W_f$, are the formulae labelling edges which connect the initial state w_0 to a final state w_n.*
- *for $w_i \in \mathit{WA}_r$ or $w_i \in \mathit{WE}_r$ for some role r, that is, w_i is an access or exit state, then $f^{\mathit{Guard}}(w_i) = \mathit{Fml}_{[\mathit{WA},i]}$ or $f^{\mathit{Guard}}(w_i) = \mathit{Fml}_{[\mathit{WE},i]}$, respectively.*
- *for $1 \leq i \leq n$, then* $\Im_i = \begin{cases} \mathbf{ext}(\mathbf{ext}(\Im_{i-1}, \mathit{Fml}_{[\mathit{WA},i]}), \mathit{Fml}_i), & \text{if } w_i \in \mathit{WA}_r \\ \mathbf{ext}(\mathbf{ext}(\Im_{i-1}, \mathit{Fml}_{[\mathit{WE},i]}), \mathit{Fml}_i), & \text{if } w_i \in \mathit{WE}_r \\ \mathbf{ext}(\Im_{i-1}, \mathit{Fml}_i), & \text{otherwise} \end{cases}$

One should notice that the existential quantification allows for the *choice* of components for the sets in Ω and hence more potential for different models. In order to obtain a model for a scene, an initial model $\Im_0$, possibly empty, must be provided.

The model of a transition extends the models of scenes connecting to it:

Definition 13. *An interpretation $\Im$ is a model for a transition $\mathbf{T} = \langle \mathit{CI}, w_a, \mathit{Fml}, w_e, \mathit{CO}\rangle$, denoted by $\mathbf{m}(\mathbf{T}, \Im)$, iff*

- *$\mathbf{S}_1, \ldots, \mathbf{S}_n$ are all the scenes that connect with CI, i.e. the set WE_i of exit states of each scene $\mathbf{S}_i, 1 \leq i \leq n$, has at least one element $\mathit{WE}_{i,r} \times w_a$ in CI, and*
- *$\mathbf{m}(\mathbf{S}_i, \Im_i), \Im_i = (\sigma_i, \Omega_i), \Im' = (\bigcup_{i=1}^{n} \sigma_i, \bigcup_{i=1}^{n} \Omega_i), 1 \leq i \leq n$, and $\mathbf{ext}(\Im', \mathit{Fml}) = \Im$*

The model of a transition is an extension of the union of the models of all its connecting scenes to accommodate *Fml*. Finally, we define the meaning of e-institutions:

Definition 14. *An interpretation $\Im$ is a model for an e-institution $\mathbf{E} = \langle \mathit{Scenes}, \mathbf{S}_0, \mathbf{S}_f, \mathit{Trans}\rangle$, denoted by $\mathbf{m}(\mathbf{E}, \Im)$, iff*

- *$\mathit{Scenes} = \{\mathbf{S}_0, \ldots, \mathbf{S}_n\}, \mathbf{m}(\mathbf{S}_i, \Im), 0 \leq i \leq n$; and*
- *$\mathit{Trans} = \{\mathbf{T}_0, \ldots, \mathbf{T}_m\}, \mathbf{m}(\mathbf{T}_j, \Im), 0 \leq j \leq m$.*

3.5 Building Models for $\mathcal{L}$-Based E-Institutions

We can build a model $\Im$ for a formula *Fml* if we are given an initial value for the sets in Ω. We need only those sets that are referred to in the quantification of *Fml*: with this information we can define the atomic formulae that make the left-hand side of "$\Rightarrow$" true. If the conditions on the left-hand side of *Fml* are fulfilled then we proceed to *make* the conditions on the right-hand side true, by means of the appropriate creation of other sets.

Building a model $\Im$ is a computationally expensive task, involving combinatorial efforts to find the atomic formulae that ought to be in σ and the contents of the sets in Ω. If, however, the formulae *Fml* of a scene have a simple property, *viz.* the quantification of each formula Fml_i only refers to sets that appear on preceding formulae $Fml_j, j < i$, then we can build an interpretation gradually, taking into account each formula at a time. This property can be syntactically checked: we can ensure that all sets appearing in Fml_i quantification appears on the right-hand side of a Fml_j which leads on to Fml_i in a scene. Only if all scenes and transitions of an e-institution fulfill this property is that we can automatically build a model for it in feasible time.

If this property holds in our e-institutions, then we can build for any formula Fml_i a model $\Im_i$ that uses the $\Im_{i-1}$ of the preceding formula (assuming an ordering among the edges of a path). The models of a scene are then built gradually, each formula at a time, via $\mathbf{ext}(\Im_{i-1}, Fml_i) = \Im_i$. The quantifiers in *Fml* assign values to variables in its body, following the semantic mapping $\mathbf{k}$ shown previously. The existential quantifiers $\exists$ and $\exists!$ introduce non-determinism: in the case of $\exists$ a subset of the elements of the quantified set has to be chosen; in the case of $\exists!$ a single element has to be chosen. Additional constraints on the choice to be made can be expressed as part of *Fml*.

Given an initial interpretation $\Im = (\emptyset, \Omega)$ in which Ω is possibly empty or may contain any initial values of sets, so that we can start building the models of the ensuing formulae. Given $\Im_{i-1}$ and Fml_i we can automatically compute the value $\mathbf{ext}(\Im_{i-1}, Fml_i) = \Im_i$. Since the quantifiers of Fml_i only refer to sets of the right-hand side of preceding Fml_j, then $\Im_{i-1}$ should have the actual contents of these sets. We exhaustively generate values for the quantified variables – this is only possible because all the sets are finite – and hence we can assemble the atomic formulae for a possible σ_i of $\Im_i$. With this σ and Ω_{i-1} we then assemble Ω_i, an extension of Ω_{i-1} which satisfies the set constraints of Fml_i.

3.6 Example: A Simple Agoric Market

We illustrate the definitions above with an example comprising a complete virtual agoric marketplace. We provide in Fig. 2 a graphic rendition of an e-institution for our market – the same e-institution is, of course, amenable to different

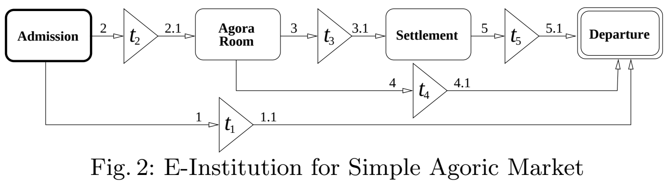

Fig. 2: E-Institution for Simple Agoric Market

visual renditions. Scenes are represented as boxes with rounded edges; the root scene **Admission** has a thicker box and the output scene **Departure** has a double box. Transitions are represented as triangles. The arcs in our diagram connect exit states of a scene with the access state of a transition and the exit state of a transition with an access state of a scene. Agents have to be initially admitted in the e-institution (**Admission** scene) where their details are recorded; agents then may proceed to trade their goods in the **Agora Room** scene, after which they may (if they have bought or sold goods) have to settle any debts in the **Settlement** scene. Finally, agents leave the institution, via the **Departure** scene.

We now focus on a specific scene in the e-institution above. In Fig. 3 we "zoom in" on the **Agora Room** scene, in which agents willing to acquire goods interact

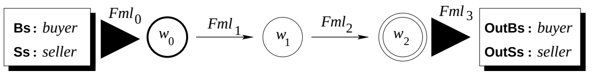

Fig. 3: Diagram for **Agora Room** Scene

with agents intending to sell such goods. This agora scene has been simplified – no auctions or negotiations are contemplated. The sellers announce the goods they want to sell and collect the replies from buyers (all buyers must reply). The simplicity of this scene is deliberate, so as to allow us to fully represent and discuss it. A more friendly visual rendition of the formal definition is employed in the figure and is explained below.

The states $W = \{w_0, w_1, w_2\}$ are displayed in circles and $\mathit{Edges} = \{(w_0, w_1), (w_1, w_2)\}$ are shown as arrows: if $(w_i, w_j) \in \mathit{Edges}$, then $w_i \longrightarrow w_j$. The initial state w_0 is shown enclosed in a thicker circle; the final state $W_f = \{w_2\}$ is enclosed in a double circle. We define the set of roles as $R = \{\mathit{seller}, \mathit{buyer}\}$. An access state $w \in \mathit{WA}$ is marked with a "▶" pointing towards the state with a box containing the role(s) of the agents that may enter the scene at that point and a set name. Exit states are also marked with a "▶" but pointing away from the state; they are also shown with a box containing the roles of the agents that may leave the scene at that point and a set name. We have defined the formulae $\mathit{Fml}_i, 0 \leq i \leq 3$, as:

$$\mathit{Fml}_0\colon \exists B, S \in \mathsf{Ags} \left(\begin{pmatrix} m(B, adm, enter(buyer)) \wedge \\ m(S, adm, enter(seller)) \end{pmatrix} \Rightarrow \begin{pmatrix} B \in \mathsf{Bs} \ \wedge \ 1 \leq |\mathsf{Bs}| \leq 10 \ \wedge \\ S \in \mathsf{Ss} \ \wedge \ 1 \leq |\mathsf{Ss}| \leq 10 \end{pmatrix}\right)$$
$$\mathit{Fml}_1\colon \forall S \in \mathsf{Ss}\, \forall B \in \mathsf{Bs}\, \exists I \in \mathsf{Is}\ (m(S, B, \mathit{offer}(I, P)) \Rightarrow \langle S, B, I, P\rangle \in \mathsf{Ofs})$$
$$\mathit{Fml}_2\colon \forall \langle S, B, I, P\rangle \in \mathsf{Ofs} \exists! A \in \mathsf{As}(m(B, S, \mathit{reply}(I, P, A)) \Rightarrow \langle B, S, I, P, A\rangle \in \mathsf{Rs})$$
$$\mathit{Fml}_3\colon \forall B \in \mathsf{Bs}\, \forall S \in \mathsf{Ss}\ \left(\begin{pmatrix} m(B, adm, leave) \wedge \\ m(S, adm, leave) \end{pmatrix} \Rightarrow \begin{pmatrix} B \in \mathsf{OutBs} \ \wedge \ S \in \mathsf{OutSs} \wedge \\ \mathsf{OutBs} = \mathsf{Bs} \ \wedge \ \mathsf{OutSs} = \mathsf{Ss} \end{pmatrix}\right)$$

The left-hand side of the Fml_i are atomic formulae which must hold in σ_i and the right-hand side are set constraints that must hold in Ω_i. The atomic formula stand for messages exchanged among the agents as they move along the edges of the scene. The above definitions give rise to the following semantics:

$$\mathbf{ext}\left(\left(\emptyset, \begin{Bmatrix} \mathsf{Ags}, \\ \mathsf{As}, \\ \mathsf{Is} \end{Bmatrix}\right), \mathit{Fml}_0\right) = \Im_0 = \left(\begin{Bmatrix} m(ag_1, adm, enter(seller)), \\ m(ag_2, adm, enter(buyer)), \\ m(ag_3, adm, enter(buyer)) \end{Bmatrix}, \begin{Bmatrix} \mathsf{Ags, As, Is,} \\ \mathsf{Bs, Ss} \end{Bmatrix}\right)$$

We assume that $\mathsf{Ags} = \{ag_1, \ldots, ag_4\}$, $\mathsf{As} = \{ok, not\_ok\}$ and $\mathsf{Is} = \{car, boat,\ plane\}$, and we obtain $\mathsf{Bs} = \{ag_2, ag_3\}$ and $\mathsf{Ss} = \{ag_1\}$;

$$\mathbf{ext}(\Im_0, \mathit{Fml}_1) = \Im_1 = \left(\sigma_0 \cup \begin{Bmatrix} m(ag_1, ag_2, \mathit{offer}(car, 4)), \\ m(ag_1, ag_3, \mathit{offer}(boat, 3)) \end{Bmatrix}, \begin{Bmatrix} \mathsf{Ags, As, Is,} \\ \mathsf{Bs, Ss, Ofs} \end{Bmatrix}\right)$$

where $\mathsf{Ofs} = \{\langle ag_1, ag_2, car, 4\rangle, \langle ag_1, ag_3, boat, 3\rangle\}$;

$$\mathbf{ext}(\Im_1, \mathit{Fml}_2) = \Im_2 = \left(\sigma_1 \cup \left\{ \begin{array}{l} m(ag_2, ag_1, reply(car, 4, ok)), \\ m(ag_3, ag_1, reply(boat, 3, not\_ok)) \end{array} \right\}, \left\{ \begin{array}{l} \mathsf{Ags, As, Is,} \\ \mathsf{Bs, Ss, Ofs,} \\ \mathsf{Rs} \end{array} \right\} \right)$$

where $\mathsf{Rs} = \{\langle ag_2, ag_1, car, 4, ok\rangle, \langle ag_3, ag_1, boat, 3, not\_ok\rangle\}$;

$$\mathbf{ext}(\Im_2, \mathit{Fml}_3) = \Im_3 = \left(\sigma_3 \cup \left\{ \begin{array}{l} m(ag_1, adm, leave), \\ m(ag_2, adm, leave), \\ m(ag_3, adm, leave) \end{array} \right\}, \left\{ \begin{array}{l} \mathsf{Ags, As, Is, Bs, Ss, Ofs,} \\ \mathsf{Rs, OutBs, OutSs} \end{array} \right\} \right)$$

where $\mathsf{OutBs} = \{ag_2, ag_3\}$ and $\mathsf{OutSs} = \{ag_1\}$.

Intuitively, the σ_i provide "snapshots" of those messages that were sent up to a particular state: the record of the messages sent characterises the state of the scene. Each state is associated with a σ_i and Ω_i. We explicitly list the messages that should be sent at each state of the scene, indicating that the complete set σ_i is an extension of σ_{i-1}. The contents of the sets in Ω_i shown above represent information relevant for defining future steps in the global protocol. This information is gathered as the protocol is followed and it defines the subsequent steps.

The semantics of transitions is defined similarly. However, the sets over which the formulae *Fml* in $\mathbf{T}_i$ are quantified are built by merging the sets of all those scenes that are connected to the transition, as formally stated in definition 13 above.

3.7 Design Rationale of $\mathcal{L}$

The class of protocols we aim at require the unambiguous reference to details of previous interactions so as to determine the ensuing message exchanges among the participating agents. In the example above, the model is gradually built taking any such restrictions into account: the quantification of the formulae labelling each edge ensures that restrictions be taken into account. For instance, Fml_2 restricts the interaction and only permits buyer agents send messages; these messages must be a reply to their respective offers. This is only possible because the semantics of $\mathcal{L}$ allows the reference to sets built to store any relevant information as edges are followed. This information is then employed via the quantifiers to restrict ensuing steps of the protocol. We are thus able to capture dynamic aspects of the protocol in a generic and abstract fashion.

The logic $\mathcal{L}$, a restricted form of first-order logic, has been engineered for our purposes of labelling connections of a finite-state machine. The set quantifications are just a notational variant of first-order quantification. It is easy to see that, for any arbitrary formula α, if $\forall X \in \mathsf{Set}.\ \alpha$ holds then $\forall X.X \in \mathsf{Set} \wedge \alpha$ also holds. The same is true for the other quantifiers $\exists$ and $\exists!$. The set constraints are just first-order predicates whose intended meaning has been "hardwired" to the underlying semantics.

There are connections between $\mathcal{L}$ and many-sorted logics [4]. The sets employed in our quantifications can be viewed as explicit sorts. However, the set constraints do not have a counterpart in many-sorted logics since sets are not part of the allowed syntax. Set-based logics are not more powerful than standard first-order logic [4]. However, we have decided to employ a set-based logic to provide for a more disciplined design with a cleaner representation. Clearly, all the sets of an $\mathcal{L}$ formula can be put together as one single set (*i.e.* the union of all sets) but if we needed to differentiate among elements (say, agents that are of different roles) then we should provide extra means. Another advantage of set-based logics stems from the potential reduction on the search space for a model: if our universe of discourse is organised in sets, our search procedure can concentrate only on the sets concerned with the formulae, thus avoiding having to unnecessarily examine large numbers of spurious elements.

3.8 Representing and Checking $\mathcal{L}$-Based E-Institutions

$\mathcal{L}$-based e-institutions can be readily represented in many different ways. We show in Fig. 4 a Prolog [1] representation for the **Agora Room** scene graphi-

```
roles(market,agora,[buyer,seller]). states(market,agora,[w0,w1,w2,w3]).
initial_state(market,agora,w0).     final_states(market,agora,[w3]).
access_states(market,agora,[buyer:[w0],seller:[w0,w2]]).
exit_states(market,agora,[buyer:[w3],seller:[w1,w3]]).
edges(market,agora,[(w0,w1),(w1,w2),(w2,w3)])
guard(market,agora,w0,[exists(B,agents),exists(S,agents)],
                      [m(B,adm,enter(buyer)),m(S,adm,enter(seller))],
                      [in(B,buyers),1 =< card(buyers) =< 10,
                       in(S,sellers),1 =< card(buyers) =< 10]).
label(market,agora,w0,w1,[forall(S,sellers),forall(B,buyers),exists(I,items)],
                         [m(S:seller,B:buyer,offer(I))],
                         [in([S,B,I],offers)]).
...
```

Fig. 4: Representation of **Agora Room** Scene

cally depicted in Fig. 3 above. Each component of the formal definition has its corresponding representation. Since many e-institutions and scenes may co-exist, the components are parameterised by the e-institution and scene names (first and second parameters, respectively). The f^{Guard} component is represented as a `guard`/6 term; to save space, we only show the first of them. Component f^{Label} is represented as `label`/7 – we only show the first of them to save space. Both `guard`/6 and `label`/7 incorporate the same representation for $\mathcal{L}$ formulae, in their last three arguments: a list for the quantifications *Qtf*, a list for the conjunction *Atfs* and a list for the set constraints *SetCtrs*. The actual coding of the logical constructs into a Prolog format is done in a simple fashion: "$\forall x \in \mathsf{Set}$" is coded as `forall(X,set)`, "$\exists x \in \mathsf{Set}$" is encoded as `exists(X,set)`, "$x \in \mathsf{Set}$" (set operation) is encoded as `in(X,set)` and so on.

The terms standing for the messages sent in the `label`s of our representation have been augmented with information on the *role* of the agents which sent them (and the roles of the agents the messages are aimed at). In our example above, the roles `seller` and `buyer` were added, respectively, to the first and second arguments of the message, that is, `m(S:seller,B:buyer,offer(I)`. We shall use this information when we automatically synthesise agents to enact our e-institutions, as explained below. This information can be inferred from the scene specification, by propagating the roles adopted by the agents which `enter`ed the scene in access states. We have adopted the standard messages `enter(`*Role*`)` and `leave` in our scenes to convey, respectively, that the agent wants to enter the scene and incorporate *Role* and that the agent wants to leave the scene.

The representation above renders itself to straightforward automatic checks for well-formedness. For instance, we can check whether all `label`/7 terms are indeed defined with elements of `states`/3, whether all `label`/6 are defined either for `access_states`/3 or `exit_states`/3, if all `access_states`/3 and `exit_states`/3 have their `guard`/6 definition, whether all pairs in `edges`/3 have a corresponding `label`/7, and so on. However, the representation is also amenable for checking important graph-related properties using standard algorithms [3]. It is useful to check, for instance, if from the state specified in `initial_state`/3 we can reach all other `states`/3, whether there are `states`/3 from which it is not possible to reach an `exit_state` (absence of *sinks*), and so on.

The use of logics for labels in our e-institutions also allows us to explore logic-theoretic issues. Given a scene, we might want to know if the protocol it describes is feasible, that is, if it is possible for a number of agents to successfully enact it. This question amounts to finding out whether there is at least one path connecting the initial (access) state to a final (exit) state, such that the conjunction of the formulae labelling its edges is satisfiable, that is, the conjunction has at least one model. Since the quantified sets in our formulae are finite then the satisfiability test for conjunctions of $\mathcal{L}$ formulae has the same complexity of propositional logic: each atomic formula with variables can be seen as a conjunction of atomic formulae in which the variables are replaced with the actual values over which they are quantified; atomic formulae without variables amount to propositions (because they can be directly mapped to $\top$ or $\bot$). There are algorithms to carry out the satisfiability test for propositional logic which will always terminate [4, 13]. Model-checking techniques (*e.g.*, [9] and [10]) come in handy here, helping engineers to cope with the exponential complexity of this problem.

Transitions are represented in a similar fashion. We show in Fig. 5 how we represented transition $\mathbf{T}_1$ of our agoric market e-institution of Fig. 2. Transition

```
access_state(market,t1,w0). exit_state(market,t1,w1).
connections_into(market,t1,[(admission,client:[w2])]).
connections_outof(market,t1,[(agora,buyer:[w0]),(agora,seller:[w0])]).
label(market,t1,w0,w1,[exists(C,registered_clients)],
                      [m(C,adm,move(agora))],
                      [in(C,agora_agents)]).
```

Fig. 5: Representation of Transition $\mathbf{T}_1$

$\mathbf{T}_1$ guarantees that only those agents that successfully registered in the **Admission** scene (their identification being included in the set `registered_clients`) and that showed their interest in joining the **Agora Room** scene (by sending the message `m(C,adm,move(agora))`) will be able to move through it. We employed the same `label`/7 construct as in the scene representation, but here it stores the *Fml* labelling the edge connecting `w0` and `w1` in the transition.

The above representation for transitions is also amenable for automatic checks. We can automatically verify, for instance, that the scenes, states and roles referred to in `connections_into`/3 and `connections_ outof`/3 are properly defined. A desirable property in transitions is that the connecting scenes have at least one model – this property, as explained above, can be automatically checked. E-institutions are collections of scenes and transitions in the format above, plus the extra components of the tuple comprising its formal definition.

3.9 Enacting $\mathcal{L}$-Based E-Institutions

We have incorporated the concepts above into a distributed enactment platform. This platform, implemented in SICStus Prolog [17], uses the semantics of our constructs to perform a simulation of an e-institution. The platform relies on a number of administrative agents, implemented as independent processes, to overlook the enactment, building models and interacting with the agents partaking the enactment via a blackboard architecture, using SICStus Linda tuple space [2, 17].

The platform starts up for each scene an administrative agent *admScene*. An initial model is available for all scenes, $\Im = (\emptyset, \Omega)$ where Ω (possibly empty) contains the values of any sets that need to be initially defined. Some of such sets are, for instance, the identity of those agents that may join the e-institution, the possible values for items and their prices, and so on. Agent *admScene* follows the edges of a scene, starting from w_0 and, using $\Im$, creates the set σ_0 of atomic formulae. Set σ_0 is assembled by evaluating the quantification of $\mathcal{L}_0$ over the sets in Ω.

An enactment of an e-institution begins with the enactment of the root scene and terminates when all agents leave the output scene. Engineers may specify whether a scene can have many instances enacted simultaneously, depending on the number and order of agents willing to enter it. We did not include this feature in our formal presentation because in logic-theoretic terms instances of a scene can be safely seen as different scenes: they are enacted independently from each other, although they all conform to the same specification.

Our platform takes into account the agents that will partake it. These are called the *performing agents* and are automatically synthesised from the description of the e-institution, as described in [19]. A performing agent sends a message by checking if the corresponding σ set contains the message it wants to send; if the message is available then the agent "sends" it by marking it as sent. This mark is for the benefit of the *admScene* agent: the *admScene* agent creates templates for *all* messages that can be sent, but not all of them may in fact be sent. The messages that have been marked as sent are those that were actually sent by the performing agents.

Similarly, a performing agent receives a messages by marking it as received. However, it can only receive a message that has been previously marked as sent by another agent. Both the sending and receiving agents use the format of the messages to ensure they conform to the format specified in the edge they are following. To ensure that an agent does not try to receive a message that has not yet been marked as sent but that may still be sent by some agent, the *admScene* agent synchronises the agents in the scene: it first lets the sending agents change state by moving along the corresponding edge, marking their messages as sent. When all sending agents have moved, then the *admScene* agent lets the receiving agents receive their messages and move to the following state of the scene.

The synchronisation among the agents of a scene is achieved via a simple semaphore represented as a term in the tuple space. The performing agents trying to send a message must wait until this semaphore has a specific value. Likewise, the agents that will receive messages are locked until the semaphore allows them to move. The performing agents inform to the *admScene* agent, via the tuple space, the state of the scene they are currently at. With this information the *admScene* agent is able to "herd" agents from one state to another, as it creates messages templates, lets the sending agents mark them as sent and then lets the receiving agents mark them as received (also retrieving their contents). Those agents that do not send nor receive can move between states without having to wait for the semaphore. All agents though synchronise at every state of the scene, that is, there is a moment in the enactment when all agents are at state w_i, then after sending and receiving (or just moving) they are all at state w_{i+1}.

Transitions are enacted in a similar fashion. The platform assigns an agent *admTrans* to look after each transition. Transitions, however, differ from scenes in two ways. Firstly, we do not allow instances of transitions. This is strictly a methodological restriction, rather than a technical one: we want transitions to work as "meeting points" for agents moving between scenes and instances of transitions could prevent this. Secondly, transitions are *permanent*, that is, their enactment never comes to an end. Scenes (or their instances), once enacted (*i.e.* all the agents have left it at an exit state), cease to exist, that is, the *admScene* agent looking after it stops.

When a scene comes to an end, the *admScene* agent records in the tuple space the model it built as a result of the scene's enactment. The atomic formulae are only important during the enactment since they actively define the interpretations being built. However, only the sets in the Ω part of the interpretation is left as a record of the enactment. This is useful for following the dynamics of the e-institution, and it is also essential for the transitions. The *admTrans* agents looking after transitions use the sets left behind by the *admScene* agents to build their models.

A model can be explicitly represented and used to guide the distributed enactment of a $\mathcal{L}$-based e-institution. The model representation should be shared by all administrative agents which would use it instead of building its own (sub-)model. Variations of an enactment can still be explored by using *partially defined* models, in which variables are allowed as part of the atfs in σ_i. For instance, σ_1 of our previous agora room scene example, could be defined as $\sigma_1 = \{m(Ag_1, Ag_2, \mathit{offer}(I_1, P_1)), m(Ag_3, Ag_4, \mathit{offer}(I_2, P_2))\} \cup \sigma_0$ that is, the actual values of the agents' identification and items/price are not relevant, but there should be *exactly* two such messages. Restrictions can be imposed or relaxed by adequately using variables or specific values.

4 Conclusions and Future Work

In this paper we have presented a formalism to represent global protocols, that is, all possible interactions among components of a multi-agent system, from a global perspective. The proposed formalism is $\mathcal{L}$ a set-based restricted kind of first-order logic that allows engineers to describe a protocol and to forge relationships among messages of one-to-one, one-to-many and many-to-many interactions.

We have put this formalism to work by embedding it within the definition of electronic institutions [5], giving rise to $\mathcal{L}$-based electronic institutions. Existing formulations of electronic institutions, *e.g.* [5, 6, 14, 19], resort to informal explanations when defining the meaning of their constructs. Our rendition, on the other hand, has its syntax and semantics formally defined using $\mathcal{L}$. We have also presented an implementation of a platform to enact e-institutions represented in our formalism. Our proposal has been exploited for rapid prototyping of large Multi-Agent Systems [20].

Our platform is a proof-of-concept prototype, engineered with two principles in mind: a minimum number of messages should be exchanged and a maximum distribution and asynchrony among processes should be achieved. Its distributed implementation allows its scale-up: more machines can be used to host its agents. Starting from an e-institution description in our formalism, represented as a sequence of Prolog constructs, the platform starts up a number of administrative agents to overlook the scenes and transitions. The same e-institution formulation is employed to synthesise the agents that will perform in the e-institution, following our approach described in [19]. The specification of the e-institution is used to guide the synthesis of the performing agents and also to control the execution of the administrative agents.

The e-institutions are represented as Prolog terms, in a declarative fashion. We have noticed that this representation is amenable for many different sorts of manipulation. We have used it, for instance, to synthesise agents [18, 19] – these are guaranteed to conform to the e-institution they were synthesised from – and also to guide the execution of general-purpose administrative agents. However, the declarative representation also allows for desirable properties to be checked before we run the e-institution, as explained before.

Our implementation does not take into account message loss or delays. We also assume that there are no malignant agents intercepting messages and impersonating other agents. Our platform can be seen as an idealised correct version of a multi-agent system to be built, whereby the performing agents stands for "proxies" of foreign heterogeneous agents, guaranteed to follow an e-institution. The practical security issues that actual heterogeneous agents are prone to are not transferred on to the e-institution platform. We are working on how agents synthesised from the e-institution specification [19] could be presented to foreign agents and customised as their proxy agents.

References

1. K. R. Apt. *From Logic Programming to Prolog.* Prentice-Hall, U.K., 1997.
2. N. Carriero and D. Gelernter. Linda in Context. *Comm. of the ACM*, 32(4):444–458, Apr. 1989.
3. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, USA, 1990.
4. H. B. Enderton. *A Mathematical Introduction to Logic.* Harcourt/Academic Press, Mass., USA, 2nd edition, 2001.
5. M. Esteva, J. Padget, and C. Sierra. Formalizing a Language for Institutions and Norms. volume 2333 of *LNAI.* Springer-Verlag, 2001.
6. M. Esteva, J.-A. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the Formal Specification of Electronic Institutions. In F. Dignum and C. Sierra, editors, *Agent Mediated E-Commerce*, volume 1991 of *LNAI.* Springer-Verlag, 2001.
7. FIPA. The Foundation for Physical Agents. `http://www.fipa.org`, 2002.
8. P. R. Halmos. *Naive Set Theory.* Van Nostrand, Princeton, New Jersey, 1960.
9. G. J. Holzmann. The SPIN Model Checker. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997.
10. G. J. Holzmann, E. Najm, and A. Serhrouchni. SPIN Model Checking: an Introduction. *Int. Journal of Software Tools for Technology Transfer*, 2(4):321–327, Mar. 2000.
11. J. Hulstijn. *Dialogue Models for Inquiry and Transaction.* PhD thesis, University of Twente, 2000.
12. Y. Labrou, T. Finin, and Y. Peng. Agent Communication Languages: the Current Landscape. *IEEE Intelligent Systems*, 14(2):45–52, 1999.

13. Z. Manna. *Mathematical Theory of Computation.* McGraw-Hill Kogakusha, Ltd., Tokio, Japan, 1974.
14. J. A. Rodríguez Aguilar, F. J. Martín, P. Noriega, P. Garcia, and C. Sierra. *Towards a Formal Specification of Complex Social Structures in Multi-Agent Systems*, pages 284–300. Number 1624 in LNAI. Springer-Verlag, Berlin, 1997.
15. P. McBurney, R. van Eijk, S. Parsons, and L. Amgoud. A Dialogue-Game Protocol for Agent Purchase Negotiations. *Journal of Autonomous Agents and Multi-Agent Systems*, 2002. In Press.
16. J. A. Rodriguez. *On the Design and Construction of Agent-mediated Electronic Institutions.* PhD thesis, Institut d'Investigació en Intel.ligència Artificial (IIIA), Consejo Superior de Investigaciones Científicas (CSIC), Spain, 2001.
17. SICS. SICStus Prolog User's Manual. Swedish Institute of Computer Science, available at `http://www.sics.se/isl/sicstus2.html#Manuals`, Feb. 2000.
18. W. W. Vasconcelos, D. Robertson, C. Sierra, M. Esteva, J. Sabater, and M. Wooldridge. Rapid Prototyping of Large Multi-Agent Systems through Logic Programming. Submitted for publication, 2003. Document available from authors upon request.
19. W. W. Vasconcelos, J. Sabater, C. Sierra, and J. Querol. Skeleton-based Agent Development for Electronic Institutions. In *Proc. 1st Int'l Joint Conf. on Autonomous Agents & Multi-Agent Systems (AAMAS 2002)*, Bologna, Italy, 2002. ACM, U.S.A.
20. W. W. Vasconcelos, C. Sierra, and M. Esteva. An Approach to Rapid Prototyping of Large Multi-Agent Systems. In *Proc. 17th IEEE Int'l Conf. on Automated Software Engineering (ASE 2002)*, Edinburgh, UK, 2002. IEEE Computer Society, U.S.A.
21. T. Wagner, B. Benyo, V. Lesser, and P. Xuan. Investigating Interactions between Agent Conversations and Agent Control Components. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, pages 314–330. Springer-Verlag: Heidelberg, Germany, 2000.