Expressive Global Protocols via Logic-Based Electronic Institutions

Wamberto W. Vasconcelos
Department of Computing Science, University of Aberdeen, Aberdeen AB24 3UE, UK
Phone: +44 (0)1224 272283, Fax: +44 (0)1224 273422
wvasconcelos@acm.org

ABSTRACT

Communication is the key feature of Multi-agent Systems. The interactions among components of a system may take many distinct forms of increasing complexity such as in auctions, negotiations and argumentations. In this paper we propose a notation to specify and reason about an expressive class of protocols. Our notation is logic-based, employing a restricted form of first-order constructs added with sets. We formally associate the syntax of our constructs with a precise semantics. The semantics enables us to automatically build explicit models which can be used to restrict the possible executions of a given Multi-Agent System. Our notation has been incorporated to a variant of electronic institutions – an encompassing and flexible approach to specify open agent organisations – and we have implemented a proof-of-concept platform to enact arbitrary protocols using our notation.

1. INTRODUCTION

A defining property of a multi-agent system (MAS) is the *communication* among its components: a MAS can be depicted by the kinds and order of messages its agents exchange [17]. We adopt the view that the design of MASs should thus start with the study of the exchange of messages, that is, the *protocols* among the agents, as explained in [19]. Such protocols are called *global* because they depict every possible interaction among all components of a MAS.

We propose a logic-based 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. As well as providing a flexible syntax for dialogues, we also formalise their semantics via the construction of models. This paper also presents a distributed implementation of a platform to enact electronic institutions specified in our formalism.

Current efforts at standardising agent communication languages like KIF and KQML [13] and FIPA-ACL [8] do not cater for dialogues: they do not offer means to represent relationships among messages. Work on dialogues (*e.g.* [12], [16] and [22]), on the other hand, prescribe the actual format, meaning and ultimate goal of the interactions. Our effort aims at providing MASs engineers with a notation for specifying interactions among the components of a MAS, 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.

The ultimate goal of our approach is to use the specification of the protocols to synthesise the individual components of a MAS [20, 21]. The kinds and order of messages exchanged among the components of the system are all explicitly represented and give rise to the actual agents that will enact the protocol. Our formalism can express many-to-many protocols (and, in particular, one-to-one, one-to-many and many-to-one interactions), allowing the specification of a broad class of interactions among agents, whether an auction, a more sophisticated negotiation or an argumentation framework.

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. In Section 4 we explain how our approach has been exploited for quickly building prototypes for MASs. Finally, in Section 5 we discuss our work, draw conclusions and give directions for future work.

2. A SET-BASED LOGIC 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 in order for agents to follow a protocol.

We aim at a broad class of protocols in which many-to-

This paper has been accepted as a poster at the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Melbourne, Australia, 14–18 July 2003.
This version is available as Technical Report **AUCS/TR0301**, Department of Computing Science, University of Aberdeen.

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:

Def. $\mathcal{L}$ consists on 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*:

Def. 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 $Terms$, where f is any functional 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 $Atfs$:

Def. 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 $Atfs$, the construct $Atf_1 \wedge Atf_2$ 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 [10]:

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

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

$MTest$ 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 $B \in \mathsf{Buyers} \wedge card(\mathsf{Buyers}) \geq 0 \wedge card(\mathsf{Buyers}) \leq 10$. We may, alternatively, employ $|\mathsf{Set}|$ to refer to the cardinality of a set, that is, $|\mathsf{Set}| = card(\mathsf{Set})$. Additionally, in order to simplify our set expressions and improve their presentation, we can use $0 \leq |\mathsf{Buyers}| \leq 10$ instead of the expression above.

Finally, we define the quantifications Qtf:

Def. The *quantification* Qtf is defined as:

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

Where $SetOp$ is any set operation, $Term$ is any term (see above) and Var is any variable from $Vars$.

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

Using the typographic conventions presented above, we can now build correct formulae; an example is

$$\exists B \in \mathsf{Ags}\,(m(B, adm, enter(buyer)) \Rightarrow (B \in \mathsf{Bs} \wedge 1 \leq |\mathsf{Bs}| \leq 10))$$

To simplify our formulae, we shall also write quantifications of the form $Qtf\ \ Var \in SetOp, Var = Term$ simply as $Qtf\ \ Term \in SetOp$. 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* for our formulae:

Def. 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 $Qtf(Atfs)$ and in Ω what is needed in order to assign a truth value to $Qtf(SetCtrs)$.

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 [5, 14]. 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 $\mathbf{k} : Fml \times \Im \mapsto \{\top, \bot\}$ is:

1. $\mathbf{k}(\forall Terms \in SetOp\ Fml, \Im) = \top$ iff
   $\mathbf{k}(Fml|_e^{Terms}, \Im) = \top$ for all $e \in \mathbf{k}'(SetOp, \Im)$
   $\mathbf{k}(\exists Terms \in SetOp\ Fml, \Im) = \top$ iff
   $\mathbf{k}(Fml|_e^{Terms}, \Im) = \top$ for some $e \in \mathbf{k}'(SetOp, \Im)$
   $\mathbf{k}(\exists! Terms \in SetOp\ Fml, \Im) = \top$ iff
   $\mathbf{k}(Fml|_e^{Terms}, \Im) = \top$ for a single $e \in \mathbf{k}'(SetOp, \Im)$
2. $\mathbf{k}((Atfs \Rightarrow SetCtrs), \Im) = \bot$ iff
   $\mathbf{k}(Atfs, \Im) = \top$ and $\mathbf{k}(SetCtrs, \Im) = \bot$
3. $\mathbf{k}(Atfs_1 \wedge Atfs_2, \Im) = \top$ iff $\mathbf{k}(Atfs_1, \Im) = \mathbf{k}(Atfs_2, \Im) = \top$
   $\mathbf{k}(Atf, \Im) = \top$ iff $Atf \in \sigma, \Im = (\sigma, \Omega)$
4. $\mathbf{k}(SetCtrs_1 \wedge SetCtrs_2, \Im) = \top$ iff
   $\mathbf{k}(SetCtrs_1, \Im) = \mathbf{k}(SetCtrs_2, \Im) = \top$
   $\mathbf{k}(Terms \in SetOp, \Im) = \top$ iff $Terms \in \mathbf{k}'(SetOp, \Im)$;
   $\mathbf{k}(Terms \notin \mathcal{O}, \Im) = \top$ iff $Terms \notin \mathbf{k}'(SetOp, \Im)$
5. $\mathbf{k}(|SetOp|\ Op\ \mathbb{N}, \Im) = \top$ iff $|\mathbf{k}'(SetOp, \Im)|\ Op\ \mathbb{N}$ holds.
   $\mathbf{k}(|SetOp_1|\ Op\ |SetOp_2|, \Im) = \top$ iff
   $|\mathbf{k}'(SetOp_1, \Im)|\ Op\ |\mathbf{k}'(SetOp_2, \Im)|$ holds
   $\mathbf{k}(SetOp_1 = SetOp_2, \Im) = \top$ iff
   $\mathbf{k}'(SetOp_1, \Im) = \mathbf{k}'(SetOp_2, \Im)$

In item 1 we address the three quantifiers over *Fml* formulae, where $Fml|_e^{Terms}$ 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 $\mathbf{k}' : SetOp \times \Im \mapsto \mathsf{Set}$ in $\Omega, \Im = (\sigma, \Omega)$, referred to above and which gives meaning to the set operations is thus defined:

1. $\mathbf{k}'(SetOp_1 \cup SetOp_2, \Im) = \{e \mid e \in \mathbf{k}'(SetOp_1, \Im) \text{ or } e \in \mathbf{k}'(SetOp_2, \Im)\}$
2. $\mathbf{k}'(SetOp_1 \cap SetOp_2, \Im) = \{e \mid e \in \mathbf{k}'(SetOp_1, \Im) \text{ and } e \in \mathbf{k}'(SetOp_2, \Im)\}$
3. $\mathbf{k}'(SetOp_1 - SetOp_2, \Im) = \{e \mid e \in \mathbf{k}'(SetOp_1, \Im) \text{ and } e \notin \mathbf{k}'(SetOp_2, \Im)\}$
4. $\mathbf{k}'(SetOp_1 \times SetOp_2, \Im) = \{(e_1, e_2) \mid e_1 \in \mathbf{k}'(SetOp_1, \Im) \text{ and } e_2 \in \mathbf{k}'(SetOp_2, \Im)\}$
5. $\mathbf{k}'((SetOp), \Im) = (\mathbf{k}'(SetOp, \Im))$.
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 [10]. 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:

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

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*:

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

3. LOGIC-BASED E-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) [6, 7, 15]. 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.

We noticed that although different formulations of e-institutions can be found in the literature [6, 7, 15, 20], 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 Figure 1 below: two agents x and y engage in a simple two-step conversation: agent x informs agent y that it wants to sell item k

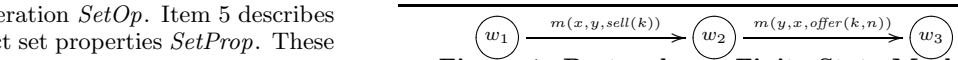

Figure 1: Protocol as a Finite-State Machine

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 Figure 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:

Def. A *scene* is $\mathbf{S} = \langle R, W, w_0, W_f, WA, WE, f^{Guard}, Edges, f^{Label} \rangle$ 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 $WA = \{WA_r \subseteq W \mid r \in R\}$ 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 [20]. 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:

Def. 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:

Def. 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 $\mathbf{T} \in Trans$, $CI \subseteq \bigcup_{i=0}^{n}(WE_i \times w_a), i \neq f$ (the exit states of the output scene can not be connected to a transition) and $CO \subseteq \bigcup_{j=1}^{n}(w_e \times WA_j)$ (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* [6] 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* [6] 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:

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

- $f^{Label}(w_{i-1}, w_i) = Fml_i, 1 \leq i \leq n, w_n \in W_f$, are the formulae labelling edges which connect the initial state w_0 to a final state w_n.
- for $w_i \in WA_r$ or $w_i \in WE_r$ for some role r, that is, w_i is an access or exit state, then $f^{Guard}(w_i) = Fml_{[WA,i]}$ or $f^{Guard}(w_i) = Fml_{[WE,i]}$, respectively.
- for $1 \leq i \leq n$, then

$$\Im_i = \begin{cases} \mathbf{ext}(\mathbf{ext}(\Im_{i-1}, Fml_{[WA,i]}), Fml_i), & \text{if } w_i \in WA_r \\ \mathbf{ext}(\mathbf{ext}(\Im_{i-1}, Fml_{[WE,i]}), Fml_i), & \text{if } w_i \in WE_r \\ \mathbf{ext}(\Im_{i-1}, Fml_i), & \text{otherwise} \end{cases}$$

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:

Def. An interpretation $\Im$ is *a model for a transition* $\mathbf{T} = \langle CI, w_a, Fml, w_e, CO\rangle$, 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 $WE_{i,r} \times w_a$ 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 $\mathbf{ext}(\Im', Fml) = \Im$

The model of a transition is an extension of the union of the models of all its connecting scenes to accommodate $\mathcal{L}$. Finally, we define the meaning of e-institutions:

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

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

3.5 Automatically Building Models

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 **k** shown previously. The existential quantifiers ∃ and ∃! introduce non-determinism: in the case of ∃ a subset of the elements of the quantified set has to be chosen; in the case of ∃! 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: An Agora Room

To illustrate the definitions above, we provide in Figure 2 a simple example of a scene for an agora room in which

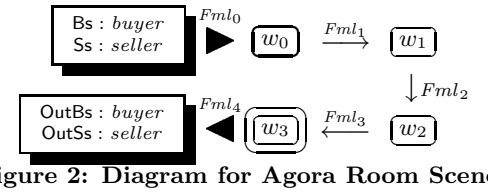

Figure 2: Diagram for Agora Room Scene

agents willing to acquire goods interact 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, collect the replies from buyers (all buyers must reply) and confirm the replies. 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 $W = \{w_0, w_1, w_2, w_3\}$ are displayed in oval boxes and $Edges = \{(w_0, w_1), (w_1, w_2), (w_2, w_3)\}$ are shown as arrows: if $(w_i, w_j) \in Edges$, then $w_i \longrightarrow w_j$. The initial state w_0 is shown enclosed in a thicker oval box; the final state $W_f = \{w_3\}$ is enclosed in a double oval box. We define the set of roles as $R = \{seller, buyer\}$. An access state $w \in WA$ 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 $Fml_i, 0 \leq i \leq 4$, as:

Fml_0: $\exists B, S \in \mathsf{Ags}$
$$\left(\begin{matrix} m(B, adm, enter(buyer)) \wedge \\ m(S, adm, enter(seller)) \end{matrix}\right) \Rightarrow \left(\begin{matrix} B \in \mathsf{Bs} \ \wedge \ 1 \leq |\mathsf{Bs}| \leq 10 \wedge \\ S \in \mathsf{Ss} \ \wedge \ 1 \leq |\mathsf{Ss}| \leq 10 \end{matrix}\right)$$

Fml_1: $\forall S \in \mathsf{Ss}\ \forall B \in \mathsf{Bs}\ \exists I \in \mathsf{Is}$
$$(m(S, B, offer(I)) \Rightarrow \langle S, B, I\rangle \in \mathsf{Ofs})$$

Fml_2: $\forall \langle S, B, I\rangle \in \mathsf{Ofs}\ \exists! A \in \mathsf{Ans}$
$$(m(B, S, reply(I, A)) \Rightarrow \langle B, S, I, A\rangle \in \mathsf{Prs})$$

Fml_3: $\forall \langle B, S, I, ok\rangle \in \mathsf{Prs}\ \exists! A \in \mathsf{Ans}$
$$(m(S, B, confirm(I, A)) \Rightarrow \langle S, B, I, A\rangle \in \mathsf{Rs})$$

Fml_4: $\forall B \in \mathsf{Bs}\ \forall S \in \mathsf{Ss}$
$$\left(\begin{matrix} m(B, adm, leave) \wedge \\ m(S, adm, leave) \end{matrix}\right) \Rightarrow \left(\begin{matrix} B \in \mathsf{OutBs} \ \wedge \ S \in \mathsf{OutSs} \wedge \\ \mathsf{OutBs} = \mathsf{Bs} \ \wedge \ \mathsf{OutSs} = \mathsf{Ss} \end{matrix}\right)$$

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:

$$\mathbf{ext}\left(\left(\emptyset, \left\{\begin{matrix} \mathsf{Ags} = \{ag_1, \ldots, ag_4\}, \\ \mathsf{Ans} = \{ok, not\_ok\}, \\ \mathsf{Is} = \{car, boat, plane\} \end{matrix}\right\}\right), Fml_0\right) = \Im_0 =$$
$$\left(\left\{\begin{matrix} m(ag_1, adm, enter(seller)) \\ m(ag_2, adm, enter(buyer)) \\ m(ag_3, adm, enter(buyer)) \end{matrix}\right\}, \left\{\begin{matrix} \mathsf{Ags}, \mathsf{Ans}, \mathsf{Is}, \\ \mathsf{Bs} = \{ag_2, ag_3\} \\ \mathsf{Ss} = \{ag_1\} \end{matrix}\right\}\right)$$

$$\mathbf{ext}(\Im_0, Fml_1) = \Im_1 =$$
$$\left(\begin{matrix} \sigma_0 \cup \\ \left\{\begin{matrix} m(ag_1, ag_2, offer(car)) \\ m(ag_1, ag_3, offer(boat)) \end{matrix}\right\} \end{matrix}, \left\{\begin{matrix} \mathsf{Ags}, \mathsf{Ans}, \mathsf{Is}, \mathsf{Bs}, \mathsf{Ss}, \\ \mathsf{Ofs} = \left\{\begin{matrix} \langle ag_1, ag_2, car\rangle, \\ \langle ag_1, ag_3, boat\rangle \end{matrix}\right\} \end{matrix}\right\}\right)$$

$$\mathbf{ext}(\Im_1, Fml_2) = \Im_2 =$$
$$\left(\begin{matrix} \sigma_1 \cup \\ \left\{\begin{matrix} m(ag_2, ag_1, reply(car, ok)) \\ m(ag_3, ag_1, offer(boat, not\_ok)) \end{matrix}\right\} \end{matrix}, \left\{\begin{matrix} \mathsf{Ags}, \mathsf{Ans}, \mathsf{Is}, \mathsf{Bs}, \mathsf{Ss}, \mathsf{Ofs}, \\ \mathsf{Prs} = \\ \left\{\begin{matrix} \langle ag_2, ag_1, car, ok\rangle, \\ \langle ag_3, ag_1, boat, not\_ok\rangle \end{matrix}\right\} \end{matrix}\right\}\right)$$

$$\mathbf{ext}(\Im_2, Fml_3) = \Im_3 =$$
$$\left(\begin{matrix} \sigma_2 \cup \\ \{m(ag_1, ag_2, confirm(car, not\_ok))\} \end{matrix}, \left\{\begin{matrix} \mathsf{Ags}, \mathsf{Ans}, \mathsf{Is}, \mathsf{Bs}, \mathsf{Ss}, \mathsf{Ofs}, \mathsf{Prs}, \\ \mathsf{Rs} = \\ \{\langle ag_1, ag_2, car, not\_ok\rangle\} \end{matrix}\right\}\right)$$

$$\mathbf{ext}(\Im_3, Fml_4) = \Im_4 =$$
$$\left(\begin{matrix} \sigma_3 \cup \\ \left\{\begin{matrix} m(ag_1, adm, leave) \\ m(ag_2, adm, leave) \\ m(ag_3, adm, leave) \end{matrix}\right\} \end{matrix}, \left\{\begin{matrix} \mathsf{Ags}, \mathsf{Ans}, \mathsf{Is}, \mathsf{Bs}, \mathsf{Ss}, \mathsf{Ofs}, \mathsf{Prs}, \mathsf{Rs}, \\ \mathsf{OutBs} = \{ag_2, ag_3\} \\ \mathsf{OutSs} = \{ag_1\} \end{matrix}\right\}\right)$$

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}.

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

We have incorporated the concepts above into a distributed enactment platform. This platform, implemented in SICStus Prolog [18], 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 [3, 18].

An administrative agent *admPlatform* overlooks the platform as a whole. A number of e-institutions can be enacted simultaneously in our platform. When an agent willing to perform in an e-institution **E** sends a message requesting access to a scene **S** from **E**, then the *admPlatform* starts up an administrative agent *admEInst* to overlook the enactment of **E**. The *admPlatform* agent ensures an *admEInst* is started up for every e-institution that agents want to enact, allowing for more decentralisation.

Our enactment platform uses a hierarchy of administrative agents, shown in Figure 3. The responsibility of the *admEinst* is to look after a specific e-institution. Such agents

$$admPlatform \left\{ \ admEInst \left\{ \begin{matrix} admScene \\ admTrans \end{matrix} \right.$$

Figure 3: Hierarchy of Administrative Agents

are generic, following the same sequence of steps for any e-institution the *admPlatform* assigns them to. The *admEInst* agent looks over the enactment of the scenes and transitions of an e-institution: for each agent that sends a message wanting to enter a scene, the *admEInst* starts up an

admScene agent to look after that scene. Likewise, for each transition of an e-institution an *admTrans* is started up to look after it.

An initial model is available for all *admScene* agents, $\Im = (\emptyset, \Omega)$ 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. The 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 e-institution description, as described in [20]. 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 those messages that can 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 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 messages at a particular edge 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 $\sigma_1 = \left\{ \begin{array}{l} m(Ag_1, Ag_2, offer(I_1)) \\ m(Ag_3, Ag_4, offer(I_2)) \end{array} \right\} \cup \sigma_0$ that is, the actual values of the agents' identification and items are not relevant, but there should be *exactly* two such messages. Restrictions can be imposed or relaxed by adequately using variables or specific values.

3.8 Checking Properties of E-Institutions

Our $\mathcal{L}$-based e-institutions have been represented as Prolog [1] terms, in a declarative fashion. We show in Figure 4 some of the terms defining the Agora Room scene depicted above. Terms are in the form `label(EInst,Scene,StateA,StateB,Label)` describing the `Label` on the edge connecting `StateA` and `StateB` in `Scene` of `EInst`. `Label` is a list of the form `[Qtf,Atfs,SetCtrs]` representing a formula of $\mathcal{L}$ in a Prolog compatible notation. Similar terms represent transitions, connecting (access and exit) states from different scenes.

```
label(market,agora,out,w0,
      [[exists(B,agents),exists(S,agents)],
       [m(B:buyer,adm,enter([market,agora,w0])),
        m(S:seller,adm,enter([market,agora,w0]))],
       (in(B,buyers) and 1 =< card(buyers) =< 10 and
        in(S,sellers) and 1 =< card(sellers) =< 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)]).
...
```

Figure 4: Prolog Representation of a Scene

A declarative representation is amenable to many different sorts of manipulation. We have used it, for instance, to synthesise agents [20] – these are guaranteed to conform to the e-institution they were synthesised from – and also to guide

the execution of general-purpose administrative agents, as depicted previously. However, the declarative representation also allows for desirable properties to be checked before we enact/run the e-institution. For instance, we can check the *well-formedness* of our e-institutions, that is, whether all scenes, transitions, roles, states, and so on, are properly defined (*e.g.* the *Frm* labels conform to the syntax), and whether scenes only connect to existing transitions and vice-versa.

More interesting properties are those concerning the accessibility of all scenes and transitions, *i.e.*, whether there is at least one path leading from an access state of the root scene to an exit state of an output scene, and the accessibility of their states, *i.e.* it is possible to reach every state in a scene from an access state. Another important property is the absence of "sinks", that is, states with no outgoing edges, and the property that the quantifiers of formulae only refer to sets assembled in preceding formulae. These properties have been formulated and checked in a straightforward way, using standard graph algorithms [4].

Another class of properties are those concerning model-theoretic issues. For instance, given an $\mathcal{L}$-based e-institution, we want to know if it is *satisfiable* [5, 14], that is, whether we can find at least one model for it. The satisfiability of an e-institution can be easily verified in an exhaustive fashion: since all our sets are finite, the quantification becomes a matter of instantiating variables to the elements of the respective sets. The worst-case scenario arises when an e-institution is *unsatisfiable* that is, we are not able to find a model for it: this test requires an exponential number of attempts, each element of every set being considered in turn.

4. RAPID PROTOTYPING VIA $\mathcal{L}$-BASED E-INSTITUTIONS

We have exploited $\mathcal{L}$-based e-institutions in an approach to rapid prototyping of large MASs [21]. Rapid prototyping offers means to explore essential features of a proposed system [11], promoting early experimentation with alternative design choices and allowing engineers to pursue different solutions without efficiency concerns [2, 9]. Our approach to prototyping MASs reflects the modelling methodology presented in [19] and consists of the following steps:

1. *Design of a Global Protocol* – in this initial step we prescribe the design of a global protocol, that is, a precise description of the kinds and order of messages that the components of the MAS can exchange via a $\mathcal{L}$-based e-institution.
2. *Synthesis and Customisation of Agents* – this step addresses the automatic synthesis of agents complying with the designed e-institution [20]. Although simple, these synthesised agents are in strict accordance with the e-institution they originate from: their behaviours conform to the specification of the global protocol. To allow for the variability of the components of a MAS and to help engineers explore the design space of individual agents, we offer means to customise the synthesised agents into more sophisticated pieces of software [21].
3. *Definition of Prototype* – a prototype consists of an $\mathcal{L}$-based e-institution and a set of corresponding customised agents. Designers may deliberately leave empty slots in the customised agents where different design possibilities may be pursued. These slots can be completed differently giving rise to distinct prototypes. A visual interface is automatically generated to enable these slots to be filled in by the designers. This interface can be seen as a console to change parameters and monitor the simulation of the prototype [21].
4. *Simulation and Monitoring of the Prototype* – the last step is the simulation of the prototype and the collection of results. For this stage we enact an e-institution using our platform explained above: agents are started up as self-contained and asynchronous processes that communicate by means of message-passing.

5. CONCLUSIONS & 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 [6], giving rise to $\mathcal{L}$-based electronic institutions. Existing formulations of electronic institutions, *e.g.* [6, 7, 15, 20], 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 [21].

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 [5]. 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 [5]. 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.

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 [20]. 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 – 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. For instance, the well-formedness of our e-institutions, that is, whether all scenes, transitions, roles, states, and so on, are properly defined, and whether scenes only connect to existing transitions and vice-versa. More interesting properties are those concerning the accessibility of scenes, transitions and states, the existence of "sinks", and the property that the quantifiers of formulae only refer to sets assembled in preceding formulae. We are currently investigating this issue.

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 [20] could be presented to foreign agents and customised as their proxy agents.

Another thread of work concerns building prototypes for MASs and studying their dynamics. We are currently investigating models for the referral of breast cancer patients and how different protocols among the parts involved (general practitioners, patients, specialists, etc.) may improve the throughput of patients with an early detection of cases.

Acknowledgements: This work was partially sponsored by the European Union, contract IST-1999-10208, research grant **Sustainable Lifecycles in Information Ecosystems** (SLIE).

6. REFERENCES

[1] K. R. Apt. *From Logic Programming to Prolog*. Prentice-Hall, U.K., 1997.

[2] Budde, R. and Kuhlenkamp, K. and Mathiassen, L. and Züllighoven, H., editor. *Approaches to Prototyping*. Springer-Verlag, New York, NY, USA, 1984.

[3] N. Carriero and D. Gelernter. Linda in Context. *Comm. of the ACM*, 32(4):444–458, Apr. 1989.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, USA, 1990.

[5] H. B. Enderton. *A Mathematical Introduction to Logic*. Harcourt/Academic Press, Mass., USA, 2nd edition, 2001.

[6] 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*, volume 2333 of *LNAI*, Berlin, 2001. Springer-Verlag.

[7] 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.

[8] FIPA. The Foundation for Physical Agents. `http://www.fipa.org`, 2002.

[9] V. S. Gordon and J. M. Bieman. Rapid Prototyping: Lessons Learned. *IEEE Software*, 12(1):85–95, 1995.

[10] P. R. Halmos. *Naive Set Theory*. Van Nostrand, Princeton, New Jersey, 1960.

[11] W. Hasselbring. Programming Languages and Systems for Prototyping Concurrent Applications. *ACM Computing Surveys*, 32(1):43–79, 2000.

[12] J. Hulstijn. *Dialogue Models for Inquiry and Transaction*. PhD thesis, University of Twente, 2000.

[13] Y. Labrou, T. Finin, and Y. Peng. Agent Communication Languages: the Current Landscape. *IEEE Intelligent Systems*, 14(2):45–52, 1999.

[14] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill Kogakusha, Ltd., Tokio, Japan, 1974.

[15] 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.

[16] 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.

[17] Michael Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, UK, Feb. 2002. ISBN 0 47149691X.

[18] SICS. SICStus Prolog User's Manual. Swedish Institute of Computer Science, available at `http://www.sics.se/isl/sicstus2.html#Manuals`, Feb. 2000.

[19] W. W. Vasconcelos, D. Robertson, J. Agustí, C. Sierra, M. Wooldridge, S. Parsons, C. Walton, and J. Sabater. A Lifecycle for Models of Large Multi-Agent Systems. In *Proc. 2nd Int'l Workshop on Agent-Oriented Soft. Eng. (AOSE-2001)*, volume 2222 of *LNCS*. Springer-Verlag, 2002.

[20] 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.

[21] 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.

[22] 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.