Model Checking Electronic Institutions

Marc-Philippe Huget[1] and **Marc Esteva**[2] and **Steve Phelps**[3] and **Carles Sierra**[4] and **Michael Wooldridge**[5]

Abstract. The design and development of *open* multiagent systems is one of the main areas in multiagent research. Human societies have successfully coped with a similar issue by creating *institutions*, which can broadly be understood as formal or semi-formal frameworks that provide commonly understood collections of rules within which people can interoperate. In ongoing work, we have been developing a framework called ISLANDER for the specification and developed of *electronic institutions*, within which software agents can meet and interoperate within a commonly understood terms of reference. This framework includes both a formal specification language for institutions and software tools to assist in their construction. In this paper, we are concerned with providing tools to support the design-time verification and validation of such institutions. We present some preliminary results in the use of a SPIN-based multiagent model checking framework called MABLE to enable verification of ISLANDER models.

1 Introduction

The design and development of open multiagent systems where a vast amount of heterogenous agents can interact is one of the main areas in multiagent research. Human societies have successfully coped with a similar issue, by creating *institutions* [13]. Institutions represent the rules of the game in a society. They define which human interactions may take place, defining what individuals are forbidden and what are permitted and under what conditions. Human institutions not only structure human interactions but also enforce individual and social behaviour by obliging everybody to act according to the norms.

Early multiagent systems research identified the advantages of organisational structuring as one of the main issues in order to cope with the complexity of designing DAI systems [6, 14, 2, 20]. In this sense, we advocate the use of electronic instituions for modelling open multiagent systems [16, 4]. Such institutions define the *participant roles*, the valid *interactions* between participants, and the *norms* that will govern them. We focus on the macro-level (societal) aspects referring to the infrastructure of electronic institutions, instead of the micro-level (internal) aspects of agents. Such a task is widely admitted by the multiagent community as highly critical [11].

Motivated by these concerns, we have developed the ISLANDER framework for electronic institutions [3, 9]. The ISLANDER framework includes a specification language for institutions with a well-defined formal semantics, and a graphical specification tool for developing ISLANDER specifications. On the one hand, ISLANDER tries to make the work of the institution designer as easy as possible, by combining textual and graphical elements for the specification, and on the other hand, it gives support to the verification of the specifications. This later point is crucial due to the complexity of these type of systems. The aim is that the tool checks the correctness of the specifications before the engineer starts the development of the infrastructure for the institution. Part of this checking involves *model checking* the institution, making use of the MABLE multiagent design and verification language [22], which is in turn implemented on top of the SPIN system for LTL model checking of finite state systems [7, 8].

The remainder of this paper is structured as follows. We begin, in the following section, by giving an overview of the ISLANDER framework and the language used to specify institutions, and go on to give a brief overview of the MABLE system. We then describe a prototype tool we have implemented, which will take as input an ISLANDER specification, and will generate as output a MABLE representation of this institution, together with formal claims about the behaviour of this system, which can be automatically checked using MABLE and SPIN. We present two short examples, illustrating the tool, and conclude by discussing some potential future research directions.

2 ISLANDER: Electronic Institution Design

The ISLANDER editor is a graphical tool developed at the IIIA as part of the E-Institutor project. The main goal of the E-Institutor project is to model *agent-mediated electronic institutions*. These institutions seem to be a convenient approach to tackle numerous heterogeneous agents in multiagent systems since they define the participant roles, the valid interactions and the norms that will govern them.

An electronic institution is composed of four elements: a dialogical framework, scenes, performatives structure and norms.The dialogical framework defines the valid illocutions that agents can exchange and which are the participant roles and their relationship. The dialogical framework provides the structure for helping heterogeneous agents to interact together thanks to a common ontology.

The activities in electronic institutions are the composition of multiple, distinct, possibly concurrent, dialogical activities, each one involving different groups of agents playing different roles. For each activity, interaction between agents are articulated through agent group meetings, also called scenes. Agents can enter or leave scenes at some particular states of the scenes and according to their role in this scene.

The performative structure specifies the connections between scenes, that is which scenes are available for the agents when they

[1] Dept. Computer Science, University of Liverpool, Liverpool, L69 7ZF, UK; mph@csc.liv.ac.uk
[2] Artificial Intelligence Research Institute, IIIA-CSIC, 08193 Bellaterra, Barcelona, Spain; marc@iiia.csic.es
[3] Dept. Computer Science, University of Liverpool, Liverpool, L69 7ZF, UK; sphelps@csc.liv.ac.uk
[4] Artificial Intelligence Research Institute, IIIA-CSIC, 08193 Bellaterra, Barcelona, Spain; sierra@iiia.csic.es
[5] Dept. Computer Science, University of Liverpool, Liverpool, L69 7ZF, UK; mjw@csc.liv.ac.uk

```
textual-specification ::= definition-list

definition-list ::= definition
                   |definition definition-list

definition ::= ins-def
              |dialog-frame-def
              |performative-def
              |ontology-def
              |scene-def
              |norm-def
              |illocution-def

ins-def ::=  (define-institution institution-id as
              dialogic-framework = dialogic-framework-id
              performative-structure = performative-structure-id
              [norms = (norm-ids)])

dialog-frame-def ::=
    (define-dialogic-framework dialogic-framework-id as
     ontology = ontology-id
     content-language = cl
     illocutionary-particles = (illocutionary-particle-ids)
     [external-roles = (ex-role-ids)]
     [internal-roles = (in-role-ids)]
     [social-structure = (social-structure-def)])

ontology-def ::=  (define-ontology ontology-id as
                   type-def-list)

type-def-list ::= type-def
                 |type-def type-def-list

type-def::= (type type-id)
           |(datatype type-id = constructor-id of type-ids)
           |(function-id : type-ids -> type-id)

performative-def ::=
  (define-performative-structure performative-structure-id as
  scenes = (scene-dec-list)
  transitions = (transition-dec-list)
  connections = (ps-conections-def-list)
  initial-scene = scene-id
  final-scene = scene-id
  )

scene-def ::=
 (define-scene scene-type-id as
  roles = (role-ids)
  scene-dialogic-framework = dialogic-framework-id
  states = (state-ids)
  initial-state = initial-state-id
  final-states = (final-state-ids)
  access-states = (role-access-states)
  exit-states = (role-exit-states)
  [agents-per-role = (min-max-def-list)]
  [connections = (sc-connection-def-list)] )

norm-def ::=
  (define-norm norm-id as
    antecedent = antecedent-def
    [defeasible-antecedent = (action-list)]
    consequent = (obligation-list)

antecedent-def ::= (action-list)
                   |((action-list) literals-list)

action-list::= (scene-id illocution-scheme)
              |(scene-id illocution-scheme) action-list

obligation-list ::= (obligued agent-var illocution-scheme scene-id)
        |(obligued agent-var illocution-scheme scene-id) obligation-list
```

Figure 1. BNF for the ISLANDER textual language.

are in a specific scene and role. Finally, norms define what is permitted, what is forbidden and what is obliged. For instance, in auction institution, agents are obliged to pay the seller the amount that they specified in any winning bid.

Scenes in ISLANDER are defined through a grammar, given in Figure 1. We can find the following information in a scene: the participant roles, the dialogic framework, the number of participants for each role and the different states of the scene. Scenes can be seen as graphs with an initial state and several final states. Moreover, each role can enter and leave the scene at different states.

We identify four basic elements of an electronic institution: dialogic framework, scenes, performative structure and norms. The dialogic framework defines the valid illocutions that agents can exchange and which are the participant roles and their relationship. In the most general case, each agent immersed in a multiagent environment is endowed with its own inner language and ontology. In order to allow agents to successfully interact with other agents we must address the fundamental issue of putting their languages and ontologies in relation. For this purpose, we propose that agents share what we call the *dialogic framework*. Institutions establish the acceptable illocutions by defining the ontology (vocabulary) —the common language to represent the "world"— and the common language for communication and knowledge representation. Moreover, the dialogic framework defines which will be the participant roles in the institution and the relationships among them. Each role defines a pattern of behaviour within the institution and any agent within an

institution is required to adopt some of them. The identification and regulation of roles is considered as part of the formalisation process of any organisation [17]. We distinguish between two types of roles: the internal and external roles. The internal roles can only be played by what we call staff agents, that is the agents that pertain to the institution. We can see them as the electronic version of the workers in human institutions. Never an external agent can play an internal role. By sharing a dialogic framework, we enable heterogeneous agents to exchange knowledge with other agents.

The activities in an electronic institution are the composition of multiple, distinct, possibly concurrent, dialogic activities, each one involving different groups of agents playing different roles. For each activity, interactions between agents are articulated through agent group meetings, which we call *scenes*, that follow well-defined communication protocols. In fact, no agent interaction within an institution takes place out of the context of a scene. We consider the protocol of each scene to model the possible dialogic interactions between roles instead of agents. In other words, scene protocols are patterns of multi-role conversation. Then, they can be multiply instantiated by different groups of agents. A distinguishing feature of scenes is that they allow agents either to enter or to leave a scene at some particular states of an ongoing conversation depending on their role.

While a scene models a particular multiagent dialogic activity, the performative structure captures more complex activities by establishing relationships among scenes. This issue arises when these conversations are embedded in a broader context, such as, for instance, organizations and institutions. If this is the case, it does make sense to capture the relationships among scenes.

In general, the activity represented by a performative structure can be depicted as a collection of multiple, concurrent scenes. Agents navigate from scene to scene constrained by the rules defining the relationships among scenes. Moreover, the very same agent can be possibly participating in multiple scenes at the same time. Likewise, there may be multiple concurrent instantiations of a scene, so we must also consider whether the agents following the arcs from one scene to another are allowed to start a new scene execution, whether they can choose to join just one or a subset of the active scenes, or even join all the active scenes. Furthermore, we may associate constraints with the arcs connecting scenes that agents must satisfy in order to traverse the arc. In order to capture the relationship between scenes we use a special type of scenes, the so-called transitions. The type of transition allows us to express agents synchronization: choose points where agents can decide which path to follow or parallelisation points where agents are sent to more than one scene. Transitions can be seen as a type of router in the context of a performative structure.

From a structural point of view, performative structures' specifications must be regarded as networks of scenes. The connections among the scenes define which agents depending on their role can move from one scene to other(s) through the defined transitions. In other words, the performative structure defines which scenes can be reached by each one of the different roles.

3 MABLE: Model Checking Multiagent Systems

MABLE is a language intended for the design and automatic verification of multiagent systems [22]. MABLE is essentially a conventional imperative programming language (similar to C), enriched by some additional constructs from the agent-oriented programming paradigm [18, 15, 10]. A MABLE system contains a number of agents, each of which is programmed using the MABLE imperative programming language. In addition, each agent in MABLE has a *mental state* consisting of "beliefs", "desires" and "intentions". An agent's beliefs intuitively correspond to the *information* that the agent has about its environment. For example, an agent might believe that the temperature of the room is 20 degrees Celsius, or that Bill Clinton is a liar. Agents can have *nested* beliefs; thus an agent might believe that Bill Clinton did not believe of himself that he was a liar. For technical reasons, we require that an agent only believes state formulae. Turning to desires, the idea is that an agent's desires represent those states of affairs that, ideally, it would like to bring about. For example, an agent might have a desire that the temperature in the room be 20 degrees Celsius, or might have a desire that Bill Clinton be impeached. As with beliefs, an agent's desires must be state formulae. An agent's intentions represent desires that the agent has committed to bring about; typically, intentions must be mutually consistent (whereas desires need not), and will persist over time [1].

```
fmla ::=
    forall IDEN ":" domain fmla      /* universal quantification */
  | exists IDEN ":" domain fmla      /* existential quantification */
  | any acceptable MABLE condition   /* primitive conditions */
  | "(" fmla ")"                     /* parentheses */
  | "[]" fmla                        /* always in the future */
  | "<>" fmla                        /* sometime in the future */
  | fmla "U" fmla                    /* until */
  | "!" fmla                         /* negation */
  | fmla "&&" fmla                   /* conjunction */
  | fmla "||" fmla                   /* disjunction */
  | fmla -> formula                  /* implication */

domain ::=
    "agent"                          /* set of all agents */
  | NUMERIC ".." NUMERIC             /* number range */
  | "{" IDEN, ..., IDEN "}"          /* a set of names */
```

Figure 2. The syntax of claims in MABLE.

Agents in MABLE are able to communicate with one-another using asynchronous message performatives in the style of the FIPA agent communication language [5]; it is in fact possible for the MABLE developer to specify the semantics of performatives, although we do not describe this mechanism here.

The MABLE language has been fully implemented. The implementation makes use of the SPIN system [7, 8], a freely available *model checking system* for finite state systems. Developed at AT&T Bell Labs, SPIN has been used to formally verify the correctness of a wide range of finite state distributed and concurrent systems, from protocols for train signalling to autonomous spacecraft control systems [8]. SPIN allows claims about a system to be expressed in propositional Linear Temporal Logic (LTL): SPIN is capable of automatically checking whether or not such claims are true or false.

The most novel aspect of MABLE is that agent definitions may be interspersed with *claims* about the behaviour of agents, expressed in $\mathcal{MORA}$, a subset of the $\mathcal{LORA}$ language introduced in [21]. $\mathcal{LORA}$ is a quantified multi-modal temporal logic, which in addition to containing the modalities of linear temporal logic (LTL) also contains modal operators for referring to the beliefs, desires, and intentions of agents. Claims can be automatically checked; in this way, we can automatically verify the behaviour of MABLE systems.

A claim is introduced outside the scope of an agent, with the keyword `claim` followed by a $\mathcal{MORA}$ formula, and terminated by a semi-colon. The formal syntax of $\mathcal{MORA}$ claims is given in Figure 2. Quantification is only allowed over finite domains, and in particular, over:

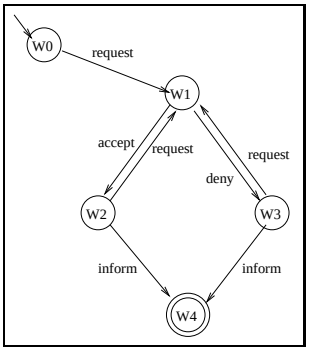

Figure 3. Graphical representation of the buyer admission scene

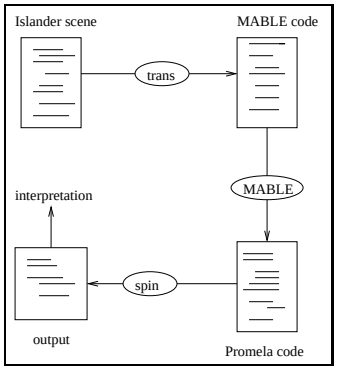

Figure 5. Model checking ISLANDER scenes

- agents (e.g., "every agent believes ϕ");
- finite sets of objects (e.g., enumeration types); and
- integer number ranges.

The MABLE compiler takes as input a MABLE system and associated claims (in $\mathcal{MORA}$) about this system. MABLE generates as output a description of the MABLE system in PROMELA (the system description language for finite-state systems used by the SPIN model checker), and a translation of the claims into the LTL form used by SPIN for model checking. The techniques used to achieve this translation, together with a more detailed description of the MABLE language, are described in [22].

Once MABLE has generated PROMELA source code and LTL claims, the SPIN model checker can be used either to automatically verify the truth (or otherwise) of the claims, or else simulate the execution of the MABLE system using the PROMELA interpreter provided as part of SPIN.

4 Using MABLE to Verify ISLANDER Scenes

In this section, we describe our approach to the automatic formal verification of ISLANDER specifications. The basic idea of the approach is as follows. An ISLANDER specification can be seen as describing a particular kind of finite state machine, with transitions in the state machine labelled with illocutionary particles. We can extract this state machine from the ISLANDER specification, and easily translate it into MABLE code. We can then express the properties that we wish to prove as claims expressed using $\mathcal{MORA}$, and run MABLE (and hence SPIN) to determine whether these properties do indeed hold. This entire procedure has been automated in a software tool called `trans`. We describe the approach in more detail, together with a description of the `trans` tool, by way of two detailed examples.

Example 1: The Buyer Admission Scene

In order to exemplify the model checking of ISLANDER scenes, we begin with one scene from the Fishmarket project and we check whether the final state is reachable. The purpose of the Fishmarket project[6] is to model a kind of auction that takes place in the fish market in the town of Blanes in Spain [12, 19].

[6] http://www.iiia.csic.es/Projects/fishmarket/newindex.html

This ISLANDER code for the *buyer admission scene* from the Fishmarket is illustrated in Figure 4. In this scene, potential buyers attempt to join an auction. This scene involves two roles: the *buyer admitter* and the *buyer*. We have only one buyer admitter for this scene but several buyers. The meaning of this scene is as follows. Buyers who want to join the scene request their admission by a *request* message (see Figure 3). This message contains the login and the password of the buyer. The buyer admitter can either accept the admission (the *accept* message) or refuse the admission (the *deny* message). In the latter case, the buyer admitter explains the reason of the deny. If buyers cannot access, they can request once again the buyer admitter. When the auction closes, the buyer admitter informs buyers and finishes the scene protocol. Figure 3 gives a graph-like representation of the scene protocol, expressed as an automaton: the initial state is indicated by an arrow entering state W0, while the final state (W4) is indicated with a double circle.

We now describe our approach to checking electronic institutions; an overview is given in Figure 5. The core of the approach is a tool called `trans`. This tool takes as input a textual representation of an ISLANDER institution (as in Figure 4). The tool generates as output:

- A MABLE representation of the institution.
- A set of basic claims defining desirable properties of the institution. By default, the claims generated simply express the fact that whatever state the scene is in, it is possible to exit cleanly from it.

The `trans` tool then invokes MABLE on the MABLE code so as to translate this MABLE code into PROMELA, which is then fed into the SPIN system for checking. The output of the verification process is then parsed, and the ISLANDER editor tool is informed of any problems.

The default claims that are checked against the system generated is that exiting "cleanly" is possible whatever state the scene is in. It is more convenient (as usual in SPIN) to look for negative examples of such claims, which we can express in the notation of LTL as:

$$\Box[(state = W_i) \rightarrow \Box\neg exit]$$

where $state$ is a variable holding the current state of the scene, W_i is a constant denoting a state, and $exit$ is a proposition that is only true when the scene has ended cleanly. This LTL formula expresses the fact that it is always the case ($\Box$) that whatever state the system is in, it is always not possible to exit. Notice we hope such claims are *false*.

```
(define-scene buyer-admission-scene as
  roles = (buyer-admitter buyer)
  scene-dialogic-framework = buyer-admission-df
  states = (W0 W4 W3 W2 W1)
  initial-state = W0
  final-states = (W4)
  acces-states = ((buyer-admitter (W0)) (buyer (W0 W3 W2)) )
  exit-states = ((buyer-admitter (W4)) (buyer (W4 W3 W2)) )
  agents-per-role = (
   (1 <= buyer-admitter <= 1)
   (buyer <= 1))
  connections = (
   (W2 W4 (inform (!y buyer-admitter) (all) (close)))
   (W0 W1 (request (?x buyer) (?y buyer-admitter) (admission (?login ?password))))
   (W3 W1 (request (?x buyer) (!y buyer-admitter) (admission (?login ?password))))
   (W3 W4 (inform (!y buyer-admitter) (all) (close)))
   (W2 W1 (request (?x buyer) (!y buyer-admitter) (admission (?login ?password))))
   (W1 W3 (deny (!y buyer-admitter) (!x buyer) (deny (?deny-code))))
   (W1 W2 (accept (!y buyer-admitter) (!x buyer) (accepted ("admission"))))
        )
)
```

Figure 4. The buyer admission scene in ISLANDER.

Expressed as MABLE claims for the buyer admission scene, we have the following:

```
claim [](scene_agent_state == W0) ->
                []!(exit_agent_scene_agent));

claim [](scene_agent_state == W1) ->
                []!(exit_agent_scene_agent));

claim [](scene_agent_state == W2) ->
                []!(exit_agent_scene_agent));

claim [](scene_agent_state == W3) ->
                []!(exit_agent_scene_agent));

claim [](scene_agent_state == W4) ->
                []!(exit_agent_scene_agent));
```

If a "claim violated" message is generated for *all* of these claims, then we know that whatever state the scene is in, it is possible to exit cleanly. For reference, the MABLE code automatically generated from the ISLANDER buyer admission scene (excluding claims) is given in Figure 6. This code only provides one agent for the scene since, at the current stage of the project, we only check the structure of the scenes and not the agents involved in the scenes.

Notice that at this point it is possible to insert additional $\mathcal{MORA}$ claims to check against the MABLE code developed. For example, we might wish to check that whenever the system is in state W2, it always subsequently enters state W4:

```
claim [](scene_agent_state == W2) ->
      <>(scene_agent_state == W4));
```

The final stage of the process is to parse the output of SPIN, and to find if the claims are violated or not. If *all* of them are violated, then we know it is possible to exit cleanly from all scene states. The `trans` tool automatically extracts this information from the SPIN output and feeds it back into the ISLANDER tool.

Example 2: Shutting Down

Our second example is a scene in which a "terminator" agent tells all agents participating in a particular scene that the scene is about to be "shut down". The scene begins with the terminator informing all agents that the application is going to close, and the participants in the scene answer that they have received the message. The terminator finally informs agents that the application is closed. We have several paths in order to consider the different combination of receiving

```
#define W4       0
#define W3       1
#define W2       2
#define W1       3
#define W0       4

int scene_agent_state;

agent scene_agent {

    scene_agent_state = W0;

    while (1) {

        switch(scene_agent_state) {
        case W3:
            choose(scene_agent_state, W1, W4);
            if (scene_agent_state == W1) {
                print("request");
            }
            if (scene_agent_state == W4) {
                print("inform");
                exit(1);
            }
        case W2:
            choose(scene_agent_state, W4, W1);
            if (scene_agent_state == W4) {
                print("inform");
                exit(1);
            }
            if (scene_agent_state == W1) {
                print("request");
            }
        case W1:
            choose(scene_agent_state, W3, W2);
            if (scene_agent_state == W3) {
                print("deny");
            }
            if (scene_agent_state == W2) {
                print("accept");
            }
        case W0:
            print("request");
            scene_agent_state = W1;
        }
    }
}
```

Figure 6. MABLE code generated for the buyer admission scene (excluding claims).

messages: `accountant`, `agora-keeper`, and `cerberus`. The ISLANDER code for this is illustrated in Figure 7.

Again, the `trans` tool processes this example and generates a number of claims in order to verify that it is possible to exit cleanly.

5 Conclusion

A key problem in multiagent systems research is that of supporting *open* environments, in which agents can join conversations arbitrarily. We have advocated the development of *electronic institu-*

```
(define-scene exitt as
  roles = (terminator cerberus accountant agora-keeper external)
  scene-dialogic-framework = exit
  states = (W9 W8 W7 W6 W5 W4 W3 W2 W1 W0)
  initial-state = W0
  final-states = (W9)
  acces-states = ((terminator (W0)) (cerberus (W1)) (accountant (W1)) (agora-keeper (W1)) (external (W1)) )
  exit-states = ((terminator (W9)) (cerberus (W8 W9)) (accountant (W8 W9)) (agora-keeper (W8 W9))) (external (W8 W9)) )
  agents-per-role = (
   (1 <= terminator <= 1)
   (cerberus <= 1)
   (accountant <= 1)
   (agora-keeper <= 1))
  connections = (
   (W1 W7 (inform (?ac accountant) (!t terminator) (gone)))
   (W6 W8 (inform (?a agora-keeper) (!t terminator) (gone)))
   (W5 W6 (inform (?ac accountant) (!t terminator) (gone)))
   (W5 W3 (inform (?a agora-keeper) (!t terminator) (gone)))
   (W1 W5 (inform (?c cerberus) (!t terminator) (gone)))
   (W4 W8 (inform (?c cerberus) (!t terminator) (gone)))
   (W3 W8 (inform (?ac accountant) (!t terminator) (gone)))
   (W2 W4 (inform (?ac accountant) (!t terminator) (gone)))
   (W2 W3 (inform (?c cerberus) (!t terminator) (gone)))
   (W1 W2 (inform (?a agora-keeper) (!t terminator) (gone)))
   (W0 W1 (inform (?t terminator) (all) (gone)))
   (W8 W9 (inform (!t terminator) (all) (closed) ?closingtime))
   (W7 W4 (inform (?a agora-keeper) (!t terminator) (gone)))
   (W7 W6 (inform (?c cerberus) (!t terminator) (gone)))
        )
)
```

Figure 7. The exitt scene in ISLANDER.

tions as a solution to this problem, and as a proof of concept of this approach, we have developed the ISLANDER framework for specifying and implementing such institutions. A key problem when designing ISLANDER institutions is to check that these designs satisfy certain desirable properties. We have developed and implemented a prototype approach to the automatic verification of ISLANDER institutions by way of the MABLE multiagent design verification language, which in turn builds upon the well-known SPIN LTL model checker. This basic approach has been validated by means of a number of worked examples.

In future work, we plan to integrate the model checking approach instigated in this report much more deeply into our general approach for developing multiagent systems. MABLE has many features for defining multiagent systems than are currently exploited in the `trans` tool. An obvious next step is to integrate the checking of institutions with the checking of multiagent systems themselves, so that it is possible to tell (for example) whether a particular multiagent system conforms to the rules of a given institution. In addition, we plan to extend the verification approach to support the checking of such institutional properties as "bidders must submit offers that are monotonically increasing in value".

Acknowledgements. This research was supported by the EC under project IST-1999-10948 (SLIE), the UK government under EPSRC project GR/R27518 (Verifiable Languages and Protocols for Multiagent Systems), and the Spanish CICYT project eINSTITUTOR (TIC2000-1414). Marc Esteva enjoys the CIRIT doctoral scholarship 1999FI-00012.

REFERENCES

[1] P. R. Cohen and H. J. Levesque, 'Intention is choice with commitment', *Artificial Intelligence*, **42**(3), 213–262, (1990).

[2] Daniel David Corkill and Victor Lesser, 'The use of meta-level control for coordination in a distributed problem solving network', in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, eds., Alan H. Bond and Les Gasser, pp. 748–756. Karlsruhe, Federal Republic of Germany, Morgan Kaufmann Publishers, (August 1983).

[3] David de la Cruz, *ISLANDER un editor d'institucions electròniques*, Master's thesis, Universitat Autonoma de Barcelona, 2001.

[4] Marc Esteva, Juan A. Rodríguez-Aguilar, Carles Sierra, Pere Garcia, and Josep L. Arcos, *Agent Mediated Electronic Commerce. The European AgentLink Perspective*, chapter On the Formal Specification of Electronic Institutions, 126–147, number 1991 in Lecture Notes in Artificial Intelligence, Springer-Verlag, 2001.

[5] FIPA, *Specification*, Foundation for Intelligent Physical Agents, http://www.fipa.org/repository/fipa2000.html, 2000.

[6] L. Gasser, C. Braganza, and N. Herman, *Distributed Artificial Intelligence*, chapter MACE: A flexible test-bed for distributed AI research, 119–152, Pitman Publishers, 1987.

[7] G. J. Holzmann, *Design and Validation of Computer Protocols*, Prentice-Hall, 1991.

[8] Gerard J. Holzmann, 'The model checker spin', *IEEE Transactions on Software Engineering*, **23**(5), (May 1997).

[9] Islander editor. http://e-institutor.iiia.csic.es/e-institutor/software/islander.html.

[10] Y. Lésperance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl, 'Foundations of a logical approach to agent programming', in *Intelligent Agents II (LNAI Volume 1037)*, eds., M. Wooldridge, J. P. Müller, and M. Tambe, 331–346, Springer-Verlag, (1996).

[11] Victor R. Lesser, 'Reflections on the nature of multi-agent coordination and its implications for an agent architecture', *Autonomous Agents and Multi-Agent Systems*, **1**, 89–111, (1998).

[12] P. Noriega, *Agent mediated auctions: The Fishmarket Metaphor*, Ph.D. dissertation, Universitat Autnoma de Barcelona, 1998.

[13] D. North, *Institutions, Institutional Change and Economics Perfomance*, Cambridge U. P., 1990.

[14] H. Edward Pattison, Daniel D. Corkill, and Victor R. Lesser, *Distributed Artificial Intelligence*, chapter Instantiating Descriptions of Organizational Structures, 59–96, Pitman Publishers, 1987.

[15] Anand S. Rao, 'Agentspeak(l): BDI agents speak out in a logical computable language', Technical Report 64, Australian Artificial Intelligence Institute, (February 1996).

[16] Juan A. Rodríguez-Aguilar, *On the Design and Construction of Agent-mediated electronic institutions*, Ph.D. dissertation, Universitat Autonoma de Barcelona, 2001. Also to appear in IIIA mongraphy series.

[17] W. R. Scott, *Organizations: Rational, Natural, and Open Systems*, Englewood Cliffs, NJ, Prentice Hall, 1992.

[18] Y. Shoham, 'Agent oriented programming', *Journal of Artificial Intelligence*, **60**, 51–92, (1993). Elsevier.

[19] C. Sierra, N. Jennings, P. Noriega, and S. Parsons, 'A framework for argumentation based negotiation', in *(ATAL97) Intelligent Agents IV*, eds., M. P. Singh, A. Rao, and M. J. Wooldridge, number 1365 in LNAI,

pp. 177–192. Springer-Verlag, (1998).
[20] Eric Werner, *Distributed Artificial Intelligence*, chapter Cooperating Agents: A Unified Theory of Communication and Social Structure, 3–36, Pitman Publishers, 1987.
[21] Michael Wooldridge, *Reasoning about Rational Agents*, MIT Press, 2000.
[22] Michael Wooldridge, Michael Fisher, Marc-Philippe Huget, and Simon Parsons, 'Model checking multi-agent systems with mable', in *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 02)*, Bologna, Italy, (July 2002).