ISLANDER: an electronic institutions editor

Marc Esteva
IIIA-CSIC Campus UAB
08193 Bellaterra, Spain
marc@iiia.csic.es

David de la Cruz
IIIA-CSIC Campus UAB
08193 Bellaterra, Spain
davdela@iiia.csic.es

Carles Sierra
IIIA-CSIC Campus UAB
08193 Bellaterra, Spain
sierra@iiia.csic.es

ABSTRACT

In this paper we present ISLANDER, a tool for the specification and verification of agent mediated electronic institutions. We have defined a textual declarative language for the specification of the components of an institution. Also an ISLANDER editor is presented. It facilitates the work of the institution designer permitting the combination of graphical and textual specifications. We take the stance that a verifiable formal specification is needed before starting the development of complex systems. This tool is our first step towards having a framework for the design and development of infrastructures for open multi-agent systems.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems, Languages and Structures; D.2.1 [**Software Engineering**]: Requirements/Specification—*Tools*

General Terms

Design, Languages

1. INTRODUCTION

The design and development of open multi-agent systems where a vast amount of heterogeneous agents can interact is one of the main areas in multi-agent research. Human societies have successfully coped with a similar issue, by creating *institutions* [7]. 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 work in DAI identified the advantages of organisational structuring as one of the main issues in order to cope with the complexity of designing DAI systems [4, 8, 1, 11]. In this sense, we advocate for modelling open multi-agent systems as electronic institutions [9, 3] which define the participant roles, the valid interactions 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 multi-agent community as highly critical [6]. This fact makes us advocate for adopting a principle engineering approach founded on a formal specification of electronic institutions, that founds their design, analysis and development of an agent architecturally-neutral approach.

Thus, in this paper we present ISLANDER [2, 5] a tool for the specification and verification of electronic institutions. On the one hand, ISLANDER tries to make as easy as possible the work of the institution designer 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 tool checks the correctness of the specifications before the engineer starts the development of the infrastructure for the institution.

This paper is structured as follows. In section 2 we introduce the fundamental concepts of an electronic institution. Then from sections 3 to 6 we explain the tool, presenting first a textual specification language, followed by the characteristics of the tool, its practical use and the hardware and software requirements for run the tool. In section 7 we present our future work and finally, in section 8 we conclude.

2. ELECTRONIC INSTITUTIONS. FUNDAMENTAL CONCEPTS

In this section we present the fundamental concepts of our model for agent mediated electronic institutions. As our goal is the development of infrastructures for electronic institutions, we focus on macro-level (societal) aspects of agents, not in their micro-level (internal) aspects. We define architecturally-neutral e-institutions, no particular agent architecture (or language) assumed for the participating agents. Thus, the designer of a participant agent is free to chose the architecture or language that is better for fulfil their goals.

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 multi-agent 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

AAMAS'02, July 15-19, 2002, Bologna, Italy.

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 [10]. 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 which are 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 moments(states) of an ongoing conversation depending on their role.

While a scene models a particular multi-agent dialogic activity, more complex activities can be specified by establishing relationships among scenes which is captured in the performative structure. This issue arises when these conversations are embedded in a broader context, such as, for instance, organisations 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 and 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 to express agents synchronisation, 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 routers 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 defines 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.

The norms which govern an organisation are one of the key sources of trust for potential participants, since they define the commitments, obligations and rights of participating agents. As described so far, the performative structure constrains the behaviour of participating agents at two levels:

- *intra-scene:* Scene protocols dictate for each agent role within a scene what can be said, by whom, to whom, and when.
- *inter-scene:* The connections between the scenes of a performative structure define the possible paths that agents may follow depending on their roles. Furthermore, the constraints over output arcs impose additional restrictions on agents attempting to reach a target scene.

These norms are, in effect, local. But, it is the agent's actions *within* a scene that may have consequences that either limit or expand its possible subsequent actions, outside the scope of the scene. The consequences we have identified take two different forms. Some actions create commitments for future actions, which may be interpreted as obligations. Other actions may affect the paths an agent may take through the performative structure because it may change which constraints are satisfied. Both types of consequences will need to be kept by an institution for each agent on an individual basis.

In order to represent the deontic notion of obligation, we set out the predicate Obl as follows:

$$Obl(x, \psi, s) = \text{agent } x \text{ is obliged to do } \psi \text{ in scene } s.$$

where ψ is taken to be an illocution scheme. We denote the set of obligations by Obl and any concrete obligation by $obl_i \in Obl$. Norms have the following schema:

$$(s_1, \gamma_1) \wedge \ldots \wedge (s_m, \gamma_m) \wedge e_1 \wedge \ldots \wedge e_n \wedge$$

$$\wedge \neg(s_{m+1}, \gamma_{m+1}) \wedge \ldots \wedge \neg(s_{m+n}, \gamma_{m+n}) \rightarrow obl_1 \wedge \ldots \wedge obl_p$$

where $(s_1, \gamma_1), \ldots, (s_{m+n}, \gamma_{m+n})$ are pairs of scenes and illocution schemes, $e_1, \ldots e_n$ are boolean expressions over illocution schemes variables, $\neg$ is a defeasible negation, and $obl_1, \ldots, obl_p$ are obligations. The meaning of these rules is that if the illocutions $(s_1, \gamma_1), \ldots, (s_m, \gamma_m)$ have been uttered, the expressions $e_1, \ldots, e_n$ are satisfied and the illocutions $(s_{m+1}, \gamma_{m+1}), \ldots, (s_{m+n}, \gamma_{m+n})$ have not been uttered, the obligations $obl_1, \ldots, obl_p$ hold. Therefore, the rules have two components, the first one is the causing of the obligations to be activated (for instance winning an auction round by saying 'mine' in a downward bidding protocol, generating the obligation to pay) and the second is the part that removes the obligations (for instance, paying the amount of money due for the round which was won).

Clearly, an external agent may not fulfil its obligations. As agents are autonomous and the institution accepts agents developed by other people, those agents cannot be forced to utter particular illocutions. Thus, it follows that the institutions cannot force agents to fulfil their obligations. However, the institution does know the obligations that each agent has acquired and can thus detect when an agent does not fulfil its obligations and hence violates the norms. Moreover, the institution can restrict the actions that an agent can carry out while it has not fulfilled some or all of its obligations.

3. ISLANDER

The first step that we did in order to have a tool for the specification of electronic institutions was to define a declarative textual language. We can see in figure 1 the BNF of the language (due to space limitations we only present the definition of the main components).

Analysing the grammar we can see that an electronic institution is specified by a dialogic framework which determines the valid illocutions and the participant roles, a performative structure which describes the activities within the institution, and a set of norms that govern the institution.

The dialogic framework contains an ontology which is a set of type and function definitions that will be used to express the content of the illocutions in the defined content language. It defines also the list of valid illocutionary particles, the set of internal and external roles and the relationships among them. This allows to express, for instance, incompatibility of roles, this is roles that can not be played by an agent at the same time.

We consider that the expressions of the communication language are constructed as formulae of the type $(\iota\,(\alpha_i\,\rho_i)\,(\beta)\,(\varphi)\,\tau)$ where ι is an *illocutionary particle*, α_i is a term which can be either an agent variable or an agent identifier, ρ_i is a term which can be either a role variable or a role identifier, β represents the adressee(s) of the message which can be an agent or a group of agents, φ is an expression of the *content language* and τ is a term which can be either a time variable or a time constant. The communication language allows to express that an illocution is addressed to an agent, to all the agents playing a role or to all the agents in the scene. If the illocution is addressed to one agent, β is of the form $\alpha_j\,\rho_j$, where α_j is a term which can be either an agent variable or an agent identifier and ρ_j is a term which can be either a role variable or a role identifier. If the illocution is addressed to all the agents of a role, β is of the form ρ_j where ρ_j is a term which can be either a role variable or a role identifier. Finally, if β is equal to the particle "all" it means that the illocution is addressed to all the agents of the scene.

As we have said the performative structure can be seen as a network of scenes. The performatifve structure determines the role flow policy between the different scenes. From the set of scenes, it must be selected which will be the initial scene that will be the enter point of the institution and the final scene which will be the exit point.

Looking to the scene definition bit we can see that in order to define an scene we need to specify which are the participant roles and its dialogic framework, that will determine the valid illocutions within the scene. A scene protocol is specified by a directed graph where the nodes represent the different states of the conversation and the arcs are labelled with illocution schemes or timeouts that make the conversation state evolve. Thus, at each point of the conversation,

```
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) bool-expr-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: Textual language BNF

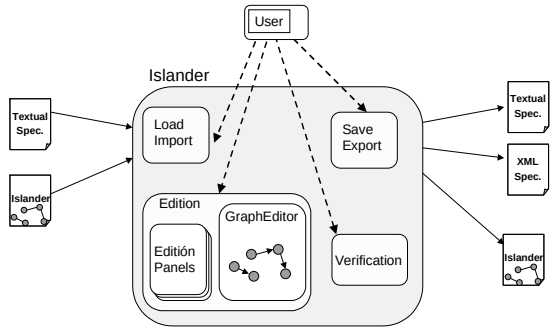

Figure 2: ISLANDER editor modules

it is defined what can be said, by whom and to whom. An important point is that when the arc is labelled with an illocution scheme some constraints on the value of the variables can be imposed. This allows, for instance, to specify that in an English auction if an agent wants to submit a bid, it has to be greater than the last submitted one.

The last element presented in figure 1 is the way to define norms that, as we have said, capture the consequences of agents actions. Norms are specified in the following form, the actions that provoke the activation of the norm expressed in the antecedent, the actions that agents must do in order to fulfil the obligations expressed in the defeasible antecedent, and the set of obligations expressed on the consequent. As we are specifying dialogical institutions, agents actions are expressed as a pair of illocution scheme and scene where it is uttered. The antecedent defines the set of illocutions that when uttered in the corresponding scene will trigger the norm making the set of obligations expressed in the consequent hold. The defeasible antecedent defines the illocutions that agent must utter in the defined scenes in order to fulfil the obligations.

This language allows to specify all the elements of an electronic institution, but we think that it would be very hard for the designer of an institution to specify all the components textually. This is specially true in the case of the protocol of a scene, the relationship among roles in the dialogic framework and the performative structure. Each one of them is represented as a graph and it is obvious that humans define and understand better a graph in its graphical representation than in its textual one. Thus, we decided to develop a tool that combines graphical and textual specification in order to facilitate the designer work. One of the important parts of the application is the graph editor that allows to specify the elements mentioned above. One of the outputs of the tool will be the specification of the institution in the so defined textual language.

4. ISLANDER EDITOR MODULES

ISLANDER editor [2] is divided in four principal modules as we can see in figure 2: Import/Load, Export/Save, Edit and Verification. Next we describe each of the modules. Also in figure 3 we can see the Data Flow Diagram of the application.

4.1 File Management

In this section we explain the Load/Import and Save/Export modules. ISLANDER editor can of course save the current specification and load previous specifications. But apart from its own format files, ISLANDER can import and export other files as we can see in figure 2. The difference is that the application files contain graphical information.

On the one hand, ISLANDER can import textual specifications. When a textual specification file is imported it goes through the verification process which informs the user of any error found. Moreover, for those elements with graphical representation this representation is automatically generated allowing the user to modify them afterwards.

ISLANDER permits to export two type of files: textual and XML. Before generating any of these type of files the specification goes through the verification process to check if it contains any error. As we will explain later, this type of files will be used in subsequent steps to develop an infrastructure of the specified institution.

4.2 Edition of electronic institutions

This module is the main module of the application. The edition module permits users to add, modify or remove all kind of elements of an institution specification. Each element is specified in a different way depending on its characteristics. Thus, ontologies, illocutions and norms are specified textually. On the other hand, dialogic frameworks, performative structures and scenes combine textual and graphical components in their specification. As in any edition process, the user interface is key in order to make as easy as possible the work of the institution designer. In section 5 we explain in detail the graphic user interface of the application.

One of the important parts of the edition is the graph editor used to specify the relationship between roles in the dialogic framework, the scene protocol and the performative structure. The graph editor has been developed independently of the rest of the application. Therefore it is used in the ISLANDER editor but could be used any other application that needs a graph editor.

In our case, the graph editor is used by the components editing the dialogic framework, the scene and the performative structure. The graph editor does not have any information of the semantics of the graph is representing and about the restrictions that this imposes on the topology of the graph. It has only information about the graph. This is, which are the nodes of the graph, the type of each node, the connections between them and the labels of the arcs. Then when the user wants to add, remove or modify any of the elements of the graph, the graph editor informs the element that it is using it and it is the element who authorises or denies the operation. The graph editor is also connected to the graphic user interface. When an operation over the graph is done the graph editor communicates it to the user interface that updates the information shown to the user.

Using this approach the graph editor can be updated without having to change the rest of the application, this is without having to modify the components using it. In our case, we can update the graph editor without having to change the components editing the dialogic framework, the scene, and the performative structure.

The rest of the elements of an institution are specified tex-

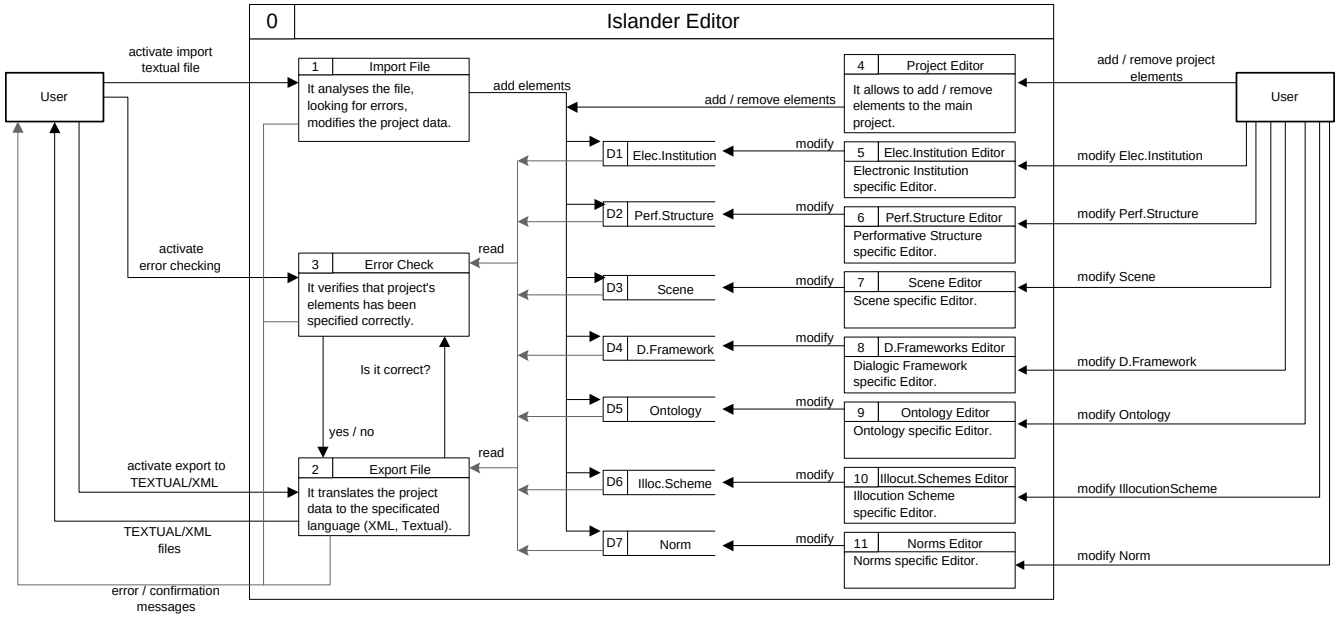

Figure 3: ISLANDER Data Flow Diagram

tually. Whenever it is possible pop-down menus are used. This is used for fields that contain references to other specification elements and for fields whose value is one of a predefined set. This facilitates the designer work because he has only to select the element from a list and it also reduces typing errors. When the user finishes the edition of a textual field, the tool checks that the definition is syntactically correct. Errors are therefore immediately detected.

4.3 Verification module

This module verifies whether the specification is correct. This is a fundamental part of the application. As open multi-agent systems are very complex it becomes crucial to be able to verify their specifications. Some verifications can be done while the user is editing the specification, but some of them need to wait until the specification is finished. Moreover this allows the user to define the elements of the institution in the order that he wants. For example, he can make references to elements not yet defined without being constantly interrupted by error messages. The user can activate the verification of the current specification whenever he wants. Also, as we have mentioned above, when he wants to import a textual specification or to export textual or XML versions of current specification the verification process is run in order to inform him of possible errors.

We have to take into account that to debug and find errors in a distributed system is a really hard work. To verify the specification and detect the errors before starting the development of the system is fundamental. This saves a lot of time and effort from the developers of the system.

In the following we concentrate on the properties verified by the tool. ISLANDER editor does the following verifications:

1. Integrity

It is obvious that the tool must check that each element which is referenced is actually defined. As we can see from the definition of the textual language some of the fields correspond to references to other specification elements. For instance, if it is defined that a scene uses a dialogic framework, this dialogic framework must be defined. Another point to check is that the dialogic framework of each one of the scenes is a subset of the general dialogic framework of the institution.

2. Liveness

One of the main properties that a multi-agent system must guarantee is that agents will not be block indefinitely. The tool checks that agents will not be block at any point of the institution and that they can always reach the final scene and leave the institution. This property must be checked at two levels, in the performative structure and in the scenes. In the performative structure for each role there must be a path from any scene that agents playing that role can reach to the output scene which represents the exit point of the institution. Moreover, we force that agents playing a role can reach all the scenes in which the role can be played by one or some of the participants, this is, from the initial scene for each role it must be a path to each of the scenes where the role can be played. All of this means that only the roles of a scene can appear in the labels of its incoming arcs and each role has to appear in at least one label of the incoming arcs. On the other hand, in order to allow agents to leave the scene, each of its roles has to appear in at least one label of its outgoing arcs. Thus agents playing the participant roles can reach and leave the scene.

We also have to guarantee that agents can entry and leave the scenes. In order to analyse the scenes we have to take into account that each scene specifies for each role a minimum and maximum number of agents that can participate playing that role and a set of access and exit states. Thus, the set of access and exit states determines the entry and exit points of the scene for a given role. When an agent wants to join a scene it has to wait until the scene reaches an entry state for its role. The same applies for the exit states when it wants to leave the scene. The tool verifies that for each

participant role there is at least one access state that allow agents playing that role to enter in the scene. Furthermore, the initial state must be an access state for those roles whose minimum is greater than zero, in order to start the scene. Also the final states must be an exit states for each one of the roles in order to allow agents to leave when the scene is finished.

3. Protocol correctness

Another important verification in the context of a scene is to check that the conversation protocol is correct. The first thing is to check that the graph representing the protocol is connected, that the initial state is not reachable once left and that there are no outgoing arcs from the final states. Then the tool analyses the labels of the arcs. In the case of illocution schemes at least, the terms referring to agents and time must be variables while the other terms can be variables or constants because we want the conversation protocol to be independent of concrete agents and time instants. Another thing to check is that the body of the message is correct with respect to the ontology of the dialogic framework of the scene. One important think is that we want the protocol to be deterministic, which means that there can not be two outgoing arcs of the same state labelled by equivalent illocutions.

The tool also analyses the use of the variables within a scene. The first thing is that each variable in a scene is correctly typed. This is, a variable can not be used first to represent, for instance, the price in an offer and later on to represent the quality of a good. We differentiate between two possible occurrences of a variable, when the variable occurrence is to be bound and when the occurrence is to consult its last bound value. It can not be the case that there exists a path within the protocol where the first occurrence of a variable is to consult its last bound value.

4. Norm correctness

Finally, the tool performs an important verification step. As we have said, actions in a norm are defined as a pair of an illocution scheme and a scene. The first point to verify is that the scene exists and that the illocution scheme labels one of its arcs. Each norm specifies the actions that will trigger it and the actions that agents must do in order to fulfil the obligations. Thus, we need to check that agents can do those actions. This means that agents can reach the scenes where they have to utter the illocutions. Therefore there must be a path in the performative structure that allows agents to move from the scenes where they have activated the rule to the scenes where they have to fulfil the obligations.

5. GRAPHIC USER INTERFACE

The graphic user interface is an essential aspect of this application because it is the way in which the institution designer interacts with the tool. In this sense the interface has been developed as user friendly as possible. We can see in figure 4 how the interface is divided in six panels that we explain next

1. Menus and tool bar.

The menus contain the general operations of the application and they are similar to other applications. In the file menu the options are: new project, close project, save, open a project, import a textual specification, export the textual representation, export the XML representation, and exit.

In the insert menu the user can insert any type of element of an institution: a performative structure, a scene, a dialogic framework, an ontology, an illocution scheme and a norm.

The tool bar contains icons for a quick access to the operations. An important one is the icon that activates the verification process.

2. Project Structure Panel.

In this panel the user can see on the upper part all the elements that belong to its current specification ordered by category. On the lower-part the sub-elements pertaining to the currently selected element are displayed. Using these two parts of the panel the user can navigate through the different elements of his specification. When he changes the selection the other panels are modified appropriately in order to show the characteristics of the selected element or sub-element.

3. Graph Panel.

The graph panel supports the edition of the graphical components of the electronic institution. The graph panel permits the edition of the graphical component of the performative structure, the dialogic framework and the scene. The user can edit the different graphs and modify them using the mouse. On the upper part there are icons that permit him to change from one graph to another. Also there is a tool bar with icons that are used to select the edition mode determining if the next action will be the selection of an element, to add a node of the selected type, or to add a connection between two nodes of the graph. As we will explain later the representation of the graph can also be modified textually from the inspect panel.

4. Textual Data Panel.

The textual data panel presents the textual representation of the current specification. It presents the textual representation of the currently selected element of the specification. We want to note that any change done by the user in the specification is immediately reflexed in the textual representation.

5. Log Panel

The log panel is used to inform the user of the errors found in the specification after the verification process is ran. Moreover, it allows the user to move to the element containing the error by simply selecting it. When a user selects an error all the panels of the application are modified in order to show him the content element containing the error.

6. Inspect Panel

This panel is used for the definition of textual components of the specification. It allows the user to modify the attributes of each one of the elements of the specification. When it is possible it uses pop-up menus to facilitate the designer work. This panel is always presented in front of the others and it always contains the information of the selected element. As soon as the user modifies an attribute of the selected element the rest of the panels are updated in order to maintain all the information consistent. When the selected element corresponds to a graph element this panel also allows to modify its graphical attributes. We can modify the position of a node, rotate it, and transform a connection to a Bezier curve.

6. HARDWARE AND SOFTWARE REQUIREMENTS

The ISLANDER editor has been developed in JAVA, version jdk1.2.2. It can run, in principle, in any machine that

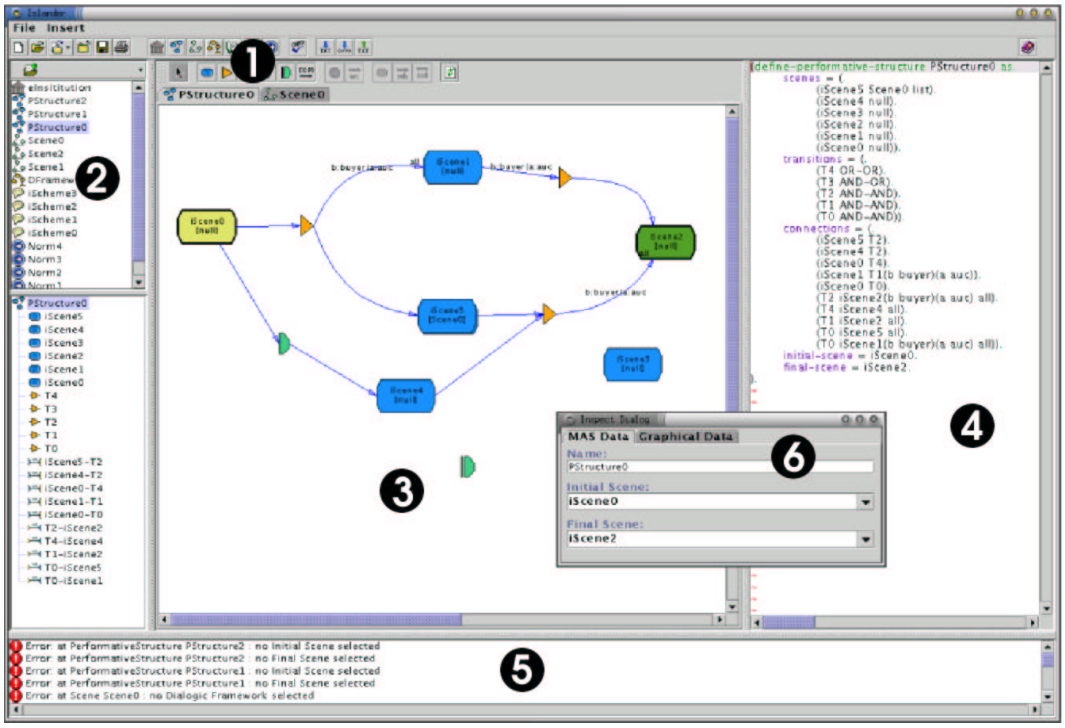

Figure 4: ISLANDER Graphic User Interface

has installed this version of JAVA. We have run the application in Linux, solaris and windows operating system. The graphic user interface consumes a lot of resources. We recommend to run it, in the case of a PC, on at least a PentiumII with 64 megabytes of RAM. The tool can be download from its web page [5].

7. FUTURE WORK

Our aim is to develop infrastructures for electronic institutions. To obtain that we want a framework that allows the specification of institutions and the subsequent development of infrastructures for the so specified institutions. In this work the tool presented here represents a first step in this direction.

We are working on two lines for the development of infrastructures for institutions. On the generation of agent templates and on a special type of agents, the so-called governors, that will mediate between the participant agents and the institution.

In the case of agent development, we advocate for the automatic extraction of agent templates from the textual specification. Concretely, we plan to generate a template for each role. We cannot generate a complete agent from the textual specification because the specification does not have enough information about how agents have to take their decisions. But from the specification we can deduce parts of the agent for each of the specified roles. Then the designer can concentrate on the important part of the agent which is the responsible of the decision making. We have to differentiate between two types of roles that lead to two types of agents, the internal and external roles. The internal roles are played for what we call the staff agents that represent the electronic equivalent of the institution workers in human institutions. The complete development of those agents is needed in order to run the institution. For this purpose we will develop a kind of agent builder that will be used by the designer to fill up the parts that can not be extracted from the textual specification. In the case of external roles the templates generated can facilitate the work of the designer that wants to develop an agent for the institution. Anyway, the designer of an external agent can always opt for developing completely his agent without using those templates.

Another fundamental element in our model are the governors. As we have said, governors will mediate between participating agents and the institution. Each participating agent is connected to a governor who mediates between it and the rest of the agents. Thus, the agent only communicates with its governor which is responsible to transmit its messages to the corresponding agents. An important function of governors is to check that agents behave according to the specification of the institution. This is, that the messages that it wants to send and the movements between different scenes are correct with respect to the specification of the institution. The governors use the specification in XML to know which are the scenes of the institution and the protocol of each one. Thus, they can evaluate whether the actions that agents want to do is correct with respect to the specification of the institution.

8. CONCLUSIONS

We think that a formal specification is needed before starting the development of complex systems. This is also true for Multi-agent systems. This formal specification process allows to identify the important parts of the system and detect possible errors saving time for the designer.

In this line we have presented ISLANDER a tool for the

specification and verification of electronic institutions. We have defined a textual language to specify institutions and the ISLANDER editor that permits the graphical specification of several language components. We think that this facilitates a lot the work of the designer because graphical specifications are extremely easy to understand. Once the specification is finished it goes through a validation process. This is fundamental to detect errors before starting the development of the infrastructure of the institution.

Acknowledgements
Marc Esteva enjoys the CIRIT doctoral scholarship 1999FI-00012. The research reported in this paper is partially supported by the european project SLIE (IST-1999-10948) and the Spanish CICYT project eINSTITUTOR (TIC2000-1414).

9. REFERENCES

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

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

[3] 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, pages 126–147. Number 1991 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 2001.

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

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

[6] 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.

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

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

[9] Juan A. Rodríguez-Aguilar. *On the Design and Construction of Agent-mediated electronic institutions*. PhD thesis, Universitat Autonoma de Barcelona, 2001. Also to appear in IIIA monography series.

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

[11] Eric Werner. *Distributed Artificial Intelligence*, chapter Cooperating Agents: A Unified Theory of Communication and Social Structure, pages 3–36. Pitman Publishers, 1987.