

ISLANDER 1.0 language definition

Marc Esteva , Carles Sierra
Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC
08193 Bellaterra, Barcelona, Spain.
{marc,sierra}@iia.csic.es
<http://www.iiia.csic.es>

Abstract

In this report we present the first version of our textual language to specify agent mediated electronic institutions where a vast amount of heterogeneous (human and software) agents play different roles and interact by means of illocutions. Thus the specification of an electronic institution will basically consist on the definition of a common language for exchanging knowledge, the activities that can be done within the institution, defined in terms of conversations, and a set of norms to capture the consequences of agents actions.

1 Introduction

Human interaction very often follows conventions, that is, general agreements on language, meaning, and behaviour. By following conventions humans decrease uncertainties about the behaviour of others, reduce conflicts of meaning, create expectations about the outcome of the interaction and simplify the decision process by restricting to a limited set the potential actions that may be taken. These benefits explain why conventions have been so widely used in many aspects of human interaction: trade, law, games, etc

On some occasions, conventions become foundational and, more importantly, some of them become norms. They establish how interactions of a certain sort will and must be structured within an organization. These conventions, or norms, become therefore the essence of what is understood as human institutions [6]. This is so for instance in the case of auction houses,

courts, parliaments or the stock exchange. Human institutions not only structure human interactions but also enforce individual and social behaviour by obliging everybody to act according to the norms.

The benefits obtained in human organizations by following conventions become even more apparent when we move into an electronic world where human interactions are mediated by computer programs, or agents. Conventions seem necessary to avoid conflicts in meaning, to structure interaction protocols, and to limit the action repertoire in a setting where the acting components, the agents, are endowed with limited rationality. Then, we advocate for modelling open multi-agent systems as electronic institutions [7, 5, 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.

Considering the computer realization of an institution, we have the view that *all* interactions among agents are realized by means of *message interchanges*. Thus, we take a strong dialogical stance in the sense that we understand a multi-agent system as a type of *dialogical system*. The interaction between agents within an institution becomes an illocution exchange. In accordance with the classic understanding of illocutions (e.g. [1] or [8]), illocutions “change the world” by establishing or modifying the *commitments* or *obligations* of the participants. This illocution exchange is structured in conversations. Thus, the activity within the institution is a set of concurrent conversations between the participants.

What a participant can do within the institution is defined by the conversations in which he can participate. This will depend on the role it is playing. Each role defines a pattern of behaviour within the institution. This is which conversations can be reach and what can he say within them. This approach of a multi-agent system defined as a set of interacting roles is also taken in [9, 4].

Due to the complexity of this type of systems, we advocate for adopting a principle engineering approach founded on a formal specification of electronic institutions, that founds their design, analysis and development. In this sense, in this report we present ISLANDER, a textual specification language for the specification of electronic institutions. The language allow for a complete and sound specification of all the components of an electronic institution. When the specification are done they will go through a validation process that will check that they are correct. This is fundamental before to start an implementation of the infrastructure for the specified institution.

2 ISLANDER

Following we present the different components of our textual language and the definition of each of them.

2.1 Electronic Institution

In this section we present the definition of electronic institution. From our point of view a dialogic electronic institution is defined by three elements: a dialogic framework, a performative structure and a set of norms.

The dialogic framework defines the language that agents will use to communicate. As an electronic institution will be populated by heterogenous agents it appears the need of defining a common language which will allow them to exchange knowledge. Moreover, the dialogic framework contains which will be the participant roles, where each role defines a pattern of behaviour within the institution.

The performative structure contains the specification of what could be done within the institution. A performative structure is a network of scenes where each scene defines a conversation protocol. Concretely, the conversation is defined in terms of roles. A scene can be multiple instantiated by different groups of agents playing its roles. Then the performative structure captures the relationships between scenes defining how agents can move between scenes depending on their role(s).

Finally the set of norms define the consequences that agents actions within scenes will have. Some actions can impose obligations on agents or can modify its possible actions within the institutions.

Thus, an electronic institution is specified as follows¹:

```
(define-institution institution-id as
  dialogic-framework = dialogic-framework-id
  performative-structure = performative-structure-id
  [norms = (norm-id+)]
)
```

In the next subsections we explain how the components presented here are defined. First we present the dialogic framework, followed by the definition of the performative structure and finally is presented the definition of norms.

¹When an element is between brackets it means that it is optional.

2.2 Dialogic Framework

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 fundamental issue of putting their languages and ontologies in relation. For this purpose, we propose that agents share what we call the *dialogic framework*, that contains the elements for the construction of the expressions of the communication language. By sharing a dialogic framework, we enable heterogeneous agents to exchange knowledge with one another.

The dialogic framework, determines the illocutions that can be exchanged between the participants. In order to do so, an ontology is defined that fixes what are the possible values for the concepts in a given domain, e.g. goods, participants, roles, locations, time intervals, etc. The dialogic framework also defines which will be the content language and the list of valid illocutionary particles.

Moreover the dialogic framework defines which are the participant roles where a role is not more than a pattern of behaviour within the institution. Thus, each participant agent will play some role(s). We differentiate between 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. Then external agents can only play external roles. external roles. Finally relations over roles can also be defined, as for instance, incompatibility of roles, this is roles that can not be played by an agent at the same time.

The dialogic framework contains the following elements:

- ontology: an identifier of a defined ontology that fixes what are the possible values for the concepts in a given domain.
- content-language: a language for the encoding of of the knowledge to be exchanged among agents using the vocabulary offered by the ontology. We are thinking in three possible content languages: PROLOG expressions, LISP expressions or KIFF expressions. But only prolog is supported at this stage.
- illocutionary-particles: a *list of names* of illocutionary particles to be used in the illocutions.
- external-roles: a *list of names* of roles that external agents may play.
- internal-roles: a *list of names* of roles that internal (staff) agents may play.

- social-structure: a triples of two *names* of roles and the *name* of relationship between them.

Then a dialogic framework is defined as follows:

```
(define-dialogic-framework dialogic-framework-id as
  [ontology = ontology-id]
  content-language = {PROLOG,LISP,KIF}
  illocutionary-particles = (illocutionary-particle-id+)
  [external-roles = (role-id+)]
  [internal-roles = (role-id+)]
  [social-structure = ((role-id relation-id role-id)+)]
)
```

Within a dialogic framework the content language allows for the encoding of the knowledge to be exchanged among agents using the vocabulary offered by the ontology. The propositions built with the aid of the content language are embedded into an “outer language”, the communication language (CL), which expresses the intentions of the utterance by means of the illocutionary particles. We take this approach in accord to speech act theory [8], which postulates that utterances are not simply propositions that are true or false, but attempts on the part of the speaker that succeed or fail.

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 addressee(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 CL 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. We say that a *CL* expression is an *illocution schema* when some of the terms contain variables. Otherwise, we say that a *CL* expression is an *illocution*. This distinction will be valuable when specifying scenes in the following section.

Variable identifiers appearing in the illocution schemes can start with either ‘?’ or ‘!’. The starting symbol will serve to differentiate when the

variable can be bound to a new value or when must be replaced by its bound value when specifying scenes. We will use unification on the semantics over variables appearing in an illocution scheme.

Next is presented how illocutions schemes to be used to label scenes arcs are specified. In this illocution schemes at least the agent and time terms are variables. The time term can be omitted on specifications.

```
(define-illocution-scheme illocution-scheme-id as
  illocutionary-particle = illocutionary-particle-id
  sender = (agent-var {role-id, role-var})
  receiver = {(agent-var {role-id, role-var}), all, role-id}
  content = CL-expr
  [time = time-var]
)
```

```
CL-expr ::= {prolog-expr, lisp-expr, KIF-expr}
agent-var ::= {?id, !id}
role-var ::= {?id, !id}
time-var ::= ?id
```

The ontology defines the vocabulary of agents will use to exchange information. It defines about what agents may talk, fixing the concepts and the possible values for them in a concrete domain. The ontology is defined by a list of external types which are defined somewhere else, a list of type definition and a list of functions. The functions returning a boolean can be considered as a predicates where the type of parameters of the function are the type of the terms of the predicate. Then, this predicates can be used in the body of the messages.

```
(define-ontology ontology-id as
  {(type type-id),
   (datatype type-name = constructor-id of type-expression),
   (function-id : type-expression)}+
)
```

2.3 Performative Structure

The notion of performative structure is perhaps the most complex since it models the relationships among scenes. In particular, we wish to highlight that although conversations (scenes) are quite widely viewed as the unit of communication between agents, limited work has been done concerning

the modelling of the relationships among different 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. Thus, while a scene models a particular multi-agent dialogic activity, more complex activities can be specified by establishing 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.

At this point we must notice that the way agents move from scene to scene depends on the type of relationship holding among the source and target scenes. Sometimes we might be interested in forcing agents to synchronise before jumping into either new or existing scene executions, or offer choice points so that an agent can decide which target scene to incorporate into, and so on. Summarising, in order to capture these relationships we consider that any performative structure contains a special type of scene, the so-called *transition* scenes², devoted to mediate different types of connections among scenes. Transitions must be regarded as a kind of routers. Therefore, instead of modelling some activity, they are intended to route agents towards their destinations in different ways, depending on the type of transition. Scenes and transitions are connected by means of *arcs*. Each scene may be connected to multiple transitions, and in turn each transition may be connected to multiple scenes. In both cases, the connection between a scene and a transition is made by means of a directed arc. Then we can refer to the source and target of each arc. And given either a scene or a transition, we shall distinguish between its incoming and outgoing arcs. Notice that there is no direct connection between two scenes, or, in other words, all connections between scenes are mediated by transitions.

The arcs connecting transitions to scenes play a fundamental role. Notice that as there might be multiple (or perhaps none) scene executions of a target scene, it should be specified whether the agents following the arcs are allowed

²Henceforth *transitions* for shorter. Although transitions are a particular class of scenes, hereafter we will be using the terms separately to distinguish non-transition scenes from transition scenes.

to start a new scene execution, whether they can choose a single or a subset of scenes to incorporate into, or whether they must enter all the available scene executions.

We define a set of different types of transitions and arcs whose semantics will highly constrain the mobility of agents among the scene instances (the ongoing activities) of a performative structure. The differences between the diverse types of transitions that we consider are based on how they allow to progress the agents that they receive towards other scenes. Let us divide each transition into two parts: the *input*, through which it receives agents from the incoming arcs, and the *output*, through which agents leave following the outgoing arcs towards other scenes. Then, the following classification of transitions is based on the behaviour that they exhibit on their input and output sides:

- **AND-AND:** They establish synchronisation and parallelism points since agents are forced to synchronise at their input to subsequently follow the outgoing arcs in parallel.
- **OR-OR:** They behave in an asynchronous way at the input (agents are not required to wait for others in order to progress through), and as choice points at the output (agents are permitted to select which outgoing arc, which path, to follow when leaving).
- **AND-OR:** They are a hybrid of the two types of transitions above: on the one hand, likewise AND-AND transitions, they force agents to synchronise on the input side, while on the other hand, likewise OR-OR transitions, they permit agents to individually make the choice of which path to follow when leaving.
- **OR-AND:** They are also a hybrid of AND-AND and OR-OR transitions. Agents are not required to wait for others on the input side, but they are forced to follow all the possible outgoing arcs.

Before stating a concrete definition of performative structure, there is a last element to be considered. Notice that although two scenes may be connected by a transition, the eventual migration of agents from a source scene instance to a target scene instance not only depends on the role of the agents but also on the results achieved by agents in previous scenes. This fact motivates the introduction of constraints over the arcs connecting scenes and transitions which are defined as a list of boolean expressions. We will require that agents satisfy the constraints, conditions, over the arc solicited to be followed when attempting to leave a scene.

The definition of a performative structure contains the following elements:

- scenes: a *list* comprising a *name* for the scene, the *class* of the scene. If there can be multiple instantiations of a scene, this will be denoted by the word ‘list’ after the class name.
- transitions: a *list* comprising a *name* for the transition and the *class* of the transition.
- connections: a *list* containing the connections from scenes to transitions and from transitions to scenes. In the first case the connection is expressed by the source scene *name*, the target transition *name*, a *list* of *pairs* of *agent-variable* and role *name*, and a *list* of constraints that will restrict agents movements. In the second case is expressed by the source transition *name*, the target scene *name*, a *list* of *pairs* of *agent-variable* and role *name*, and a *name* defining if a new execution of the scene will be created or if the agent(s) will go to one, some or all current executions.
- initial-scene: the *name* of the initial scene – from one of those given in **scenes**.
- final-scene: the *name* of the final scene – from one of those given in **scenes**.

Then a performative structure is defined as follows:

```
(define-performative-structure performative-structure-id as
  scenes = ((scene-id scene-type-id [list]))+)
  transitions = ((transition-id transition-type-id)+)
  connections =
    ({(scene-id transition-id ((agent-var role-id)+ [(bool-expr)]),
      (transition-id scene-id ((agent-var role-id)* destination))})+
  initial-scene = scene-id
  final-scene = scene-id
)
```

```
transition-type ::= {AND-AND,OR-OR,AND-OR,OR-AND}
```

```
destination ::= {new,one,some,all}
```

Next we will present the definition of a scene.

2.4 Scene

We begin by explaining precisely what, for our needs, we regard to be the purpose of a scene. 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.

A scene protocol is specified by a directed graph where the nodes represent the different states of the conversation and the directed arcs are labelled with illocution schemes or timeouts. The graph has a single initial state (unreachable once left) and a set of final states representing the different endings of the conversation. There is no directed arc connecting a final state to some other state.

The information interchanged between agents is expressed in the form of illocutions. In order for the protocol to be generic some details have to be abstracted, for instance, which concrete agents are playing it. This means that state transitions cannot be labelled by grounded illocutions, instead illocutions schemes have to be used, where, at least, the terms referring to agents and time must be variables while the other terms can be variables or constants. As mentioned above, we want the conversation protocols to be generic, that is, independent of concrete agents and time instants.

Normally, the correct evolution of a conversation protocol requires a certain number of agents to be present for each role participating in the protocol. Thus, we define a minimum and a maximum number of agents for each role and this constraint is enforced by the institution. However, because we aim at modelling multi-agent conversations whose set of participants may dynamically vary, scenes allow for agents either joining or leaving at particular points during a conversation. We specify this by identifying, for each role, a set of access and exit states. The final state(s) appear in the exit states of each role, to permit all the agents to leave when the scene is finished. The initial state must be in the set of access states for those roles whose minimum is greater than zero, since this constraint implies that an agent playing that role is required in order to start the scene.

Recall that the arcs connecting the states of a scene are labelled with illocution schemes containing variables. During a scene conversation, the variables in illocution schemes are bound to the values of the actually uttered illocutions. In order to verify the correctness of subsequent illocutions we need to keep these bindings because they may restrict the values of the arguments of subsequent illocutions. These bindings change dynamically, that is, the same variable may be bound to different values at different stages

of the conversation within a scene.

For instance, in a negotiation scene between two agents we are interested to refer to the last proposal of each agent. To do this we use the same variable for all transitions in the protocol referring to our opponent proposal and added to the environment. We need to distinguish when the variable occurrence is to be bound and when the variable occurrence is to consult its currently bounded value. For instance, in a protocol for iterated negotiation, when an agent makes a new proposal a new variable binding is created and when an agent accepts the last proposal of the other agent we want the variable in the accept illocution to be the same as the value of its last binding. In order to model this we use two different prefixing symbols of the variable identifier. When the variable is preceded by a ‘?’ it is a binding occurrence, when the variable is preceded by a ‘!’ it is an application occurrence (we refer to its value). An application occurrence must be preceded by at least one binding occurrence for the protocol to be correct; at the beginning of the scene all variables are unbound.

The utterance of an actual illocution during the conversation will be matched against the illocutions schemes outgoing the current state. This matching will generate a substitution $\sigma_{w_i w_j}$ for those variables in the scheme prefixed by ‘?’, where w_i and w_j stands for the source and target state of the transition. For instance, the submission of a bid by an agent *John* playing the buyer role in an auction scene by means of the illocution (*commit(John buyer)(James auctioneer)(bid* matching the illocution scheme (*commit(?y buyer)(!x auctioneer)(bid(!good_id, ?offer))*)) between the states w_i and w_j , will generate the substitution $\sigma_{w_i w_j} = [?y/John, ?offer/25]$. Thus, we can define Σ as the sequence of all the substitutions done during a conversation, i.e. a sequence of $\sigma_{w_i w_j}$ each one corresponding to an actually uttered illocution.

Thus, a variable prefixed by ‘!’ denotes the last bound value of the variable. This is easy to compute to look backward in Σ for the first substitution in which the variable appears. Moreover as we have in Σ all the bindings established during the conversation we can obtain a subset or all the bindings for a concrete variable. This will be valuable, as we will see in next section, when specifying constraints.

Now we explain the use of constraints to model how past illocutions affect a scene’s future evolution. In practical terms, constraints will be used to restrict the set of values to create new bindings of the variables in the illocution schemes, as well as the paths that a scene conversation can follow. Sometimes the constraints can completely fix a concrete value and sometimes they just restrict the set of possible values. For instance, in an auction scene making use of the english protocol the value of any bid has to be higher than the last submitted bid, or when the auctioneer utters the illocution declaring

the winner, the value of the variable representing the winner has to be the identifier of the agent who has submitted the highest bid.

As we said we store in Σ the sequence of substitutions produced during the conversation. Prefix ‘!’ allows to obtain the last binding for a concrete variable. But, in some occasions we need to access to several past bindings of a variable. For instance, to check that the auctioneer announces the correct winner and price for a bidding round. To do so we need to recover the bindings for each submitted bid in the last round and look which is the highest bid and the buyer who has submitted this bid. The expression $(!w_i w_j x)$ will return *all* the bindings for variable x in the substitutions generated by the illocutions that lead from state w_i to w_j including loops over w_i , w_j and any intermediary state. To compute this we simply search backwards in Σ looking for the first appearance of w_j and returning all the bindings for x from there and until the first transition to w_i from another state. Moreover we may be interested in obtaining the bindings of the last two times that the conversation evolved from w_i to w_j . We note that with: $(!w_i w_j 2 var_id)$. In general, $(!w_i w_j n var_id)$. Also we permit to write conditions that restrict the set of returned bindings.

We here summarise the meaning of the different prefixes of a variable identifier:

- $?x$: stands for an occurrence of variable x that will be bound to its instantiation value. Σ will be extended after the binding is produced.
- $!x$: stands for the last binding of variable x .
- $(!w_i w_j i x)$: stands for the n -tuple of all the bindings of variable x in the i last subdialogues between w_i and w_j . $!w_i w_j 1 x$ is noted as $!w_i w_j x$ for simplicity.
- $(!w_i w_j * x)$: stands for the n -tuple of the bindings of variable x in all subdialogues between w_i and w_j in Σ .
- $(!w_i w_j i x (cond))$: stands for the n -tuple of all the bindings of variable x in the i last between w_i and w_j such that the substitution σ where the binding appears satisfies *cond*. The condition *cond* can be a list of variable identifiers that must appear on the substitution for selecting x binding or a boolean expression.

Constraints have this form: $(op expr_i expr_j)$. The expressions must be of the following basic types: string, numeric and boolean, or a multiset³ of any of these types, and where the operations are:

³A multiset is a set where elements can be repeated

- $=, ! =, <, <=, >=, > : numeric \times numeric \rightarrow boolean$
 - $=, ! = : string \times string \rightarrow boolean$
 - $=, ! = : boolean \times boolean \rightarrow boolean$
 - $in, notin : \alpha \times \alpha multiset \rightarrow boolean$, where α is any basic type.
 - $subset, subseteq : \alpha multiset \times \alpha multiset \rightarrow boolean$,
 - $or : boolean \times boolean \rightarrow boolean$
- where α is any basic type.

In conclusion, for any illocution uttered by an agent to be valid it has to match an illocution scheme of an outgoing arc of the current state, it has to respect the bindings of bound variables, and it has to satisfy the constraints.

- roles: a *list* of *names* of role that may participate in the scene.
- dialogic-framework: the *name* of the dialogic framework to be used for communication within the scene.
- states: a *list* of the *names* of the states of the conversation graph.
- initial-state: a *name* identifying the initial state.
- final-states: a *list* of *names* identifying final states.
- access-states: a *list* of *pairs* of role *name* and a *list* of states, identifying which roles may join at which states.
- exit-states: a *list* of *pairs* of role *name* and a *list* of states, identifying which roles may leave at which states.
- agents-per-role: a *list* of *triples* of role *name*, minimum *integer* and maximum *integer*, defining the constraints on the population of a particular role. If nothing is said about the minimum or maximum for a concrete role, it is understood that the minimum is zero and that there is no maximum for this role.
- connections: a *list* of the transitions between scene states. Each one comprises one or a list of source state *names*, a succeeding state *name*, and either an *illocution-scheme* with a list constraints over scenes variables, expressed by an operator and two expressions, which must be satisfied to progress through this transition or a timeout that will trigger the transition when will expire.

```

(define-scene scene-type-id as
  roles = (role-id+)
  scene-dialogic-framework = dialogic-framework-id
  states = (state-id+)
  initial-state = state-id
  final-states = (state-id+)
  acces-states = ((role (state-id+))+)
  exit-states = ((role (state-id+))+)
  [agents-per-role = (([min <=] role-id [<= max]))+]
  [connections =
    (({state-id,(state-id+)} state-id {illocution-scheme
      ((op expr expr)*), time-out}))+]
)

illocution-scheme ::=
  {illocution-scheme-id,
    (illocutionary-particle-id (agent-var {role-id, role-var})
      {(agent-var {role-id, role-var}), all, role-id}
      CL-expr [time-var])}

op ::= { <, >, !=, <=, >=, <>, in, notin, subset, subseteq, or}

```

2.5 Norms

The norms which govern an organization 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 *obliged* 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 O and any concrete obligation by $o_i \in O$.

Norms must define the actions that will provoke the activation of the norm, the obligations that agents will have and the actions that agents must carry out in order to fulfil the obligations. As we are specifying dialogical institutions, agents actions are expressed as a pair of illocution scheme and scene where it is uttered. We need both components because the same illocution could appear in more than one scene. The scene gives the context in which the illocution must be interpreted and of course, this affects the consequences that the utterance of the illocution has. That is to say, the same illocution may have different consequences in different scenes because it is uttered in a different context. Next we introduce some specific rules, the so-called *norms*, to capture which agent actions (illocutions) have consequences that need to be recorded. Norms have the following schema

$$(s_1, \gamma_1) \wedge \dots \wedge (s_m, \gamma_m) \wedge e_1 \wedge \dots \wedge e_n \wedge \\ \wedge \neg(s_{m+1}, \gamma_{m+1}) \wedge \dots \wedge \neg(s_{m+n}, \gamma_{m+n}) \rightarrow obl_1 \wedge \dots \wedge obl_p$$

where $(s_1, \gamma_1), \dots, (s_{m+n}, \gamma_{m+n})$ are pairs of scenes and illocution schemes, e_1, \dots, e_n are boolean expressions over illocution schemes variables, \neg is a defeasible negation, and obl_1, \dots, obl_p are obligations. The meaning of these rules is that if the illocutions $(s_1, \gamma_1), \dots, (s_m, \gamma_m)$ have been uttered, the expressions e_1, \dots, e_n are satisfied and the illocutions $(s_{m+1}, \gamma_{m+1}), \dots, (s_{m+n}, \gamma_{m+n})$ have *not* been uttered, the obligations obl_1, \dots, 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 downwards 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).

Then the definition of a norm contains:

- antecedent: a *list* comprising an arbitrary number of *pairs* of scene *name* and illocution-scheme *name*.

- defeasible-antecedent: a *list* comprising an arbitrary number of *pairs* of scene *name* and illocution-scheme *name*.
- consequent: a *list* of obliged *predicates* or an expression denoting the consequence of the actions made or not made.

Norms are defined in Islander as follows:

```
(define-norm norm-id as
  antecedent = {((scene-id illocution-scheme)+),
                ((scene-id illocution-scheme)+) ((bool-expr)+)}
  defeasible-antecedent = ((scene-id illocution-scheme)+)
  consequent = ((obliged agent-var illocution-scheme scene-id)+)
)
```

3 Conclusions and Future Work

In this report we have presented ISLANDER a textual specification language for electronic institutions. We have shown how all the components of an electronic institution are specified. Notice that the specification of an electronic institution must be regarded as a compositional activity to be undertaken by the institution designer. We can consider that specifications of dialogic frameworks, scenes, and performative structures are to be naturally organised into specification libraries that may be reused for composing new specifications.

In order to facilitate the designer work, we are developing an ISLANDER editor[2] a specification and verification tool for electronic institutions. The tool permits the complete specification of an electronic institution but it allows when possible to specify the elements graphically. Graphical specifications facilitate the designer work and they are also easy to understand. One of the outputs of the tool will be the specified institution on the presented textual language. Moreover, the tool will also verify the correctness of the specifications. Due to the complexity of this type of systems is crucial to verify the specifications before start the development of infrastructures for them.

Our objective is to develop social infrastructures for the specified institutions. The social infrastructure will be in charge of allowing agents interactions but controlling that participating agents do not violate the institution rules. This is to check that the actions that agents try to do within the institution are correct respect the institution specification. Then the agents composing the social infrastructure will load the institution specification.

We are also working in the development of agents for electronic institutions. From the institution specification cannot be generate a complete agent because the specification does not have enough information about how agents have to take their decisions. But agent templates can be automatic extracted from the textual specification. Then the designer can concentrate on the important part of the agent which is the responsible of the decision making. In orther to help the dessigner 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. Anyway, the designer of an external agent can always opt for developing completely his agent without using those templates.

References

- [1] J. L. Austin. *How to Do Things With Words*. Oxford University Press, 1962.
- [2] David de la Cruz. *Islander un editor d'institucions electròniques*. Master's thesis, Universitat Autònoma de Barcelona, 2001.
- [3] Marc Esteva, Juan A. Rodríguez-Aguilar, Carles Sierra, Josep L. Arcos, and Pere Garcia. *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] Jacques Ferber and Olivier Gutknetch. A meta-model for the analysis of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 128–135, 1998.
- [5] Pablo Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor*. Number 8 in IIIA Monograph Series. Institut d'Investigació en Intel·ligència Artificial (IIIA), 1997. PhD Thesis.
- [6] D. North. *Institutions, Institutional Change and Economics Perfomance*. Cambridge U. P., 1990.
- [7] Juan A. Rodríguez-Aguilar. *On the Design and Construction of Agent-mediated Electronic Institutions*. PhD thesis, Universitat Autònoma de Barcelona, 2001. Also to appear in IIIA monography series.
- [8] J. R. Searle. *Speech acts*. Cambridge U.P., 1969.

- [9] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (AGENTS'99)*, May 1999.