An Integrated Development Environment for Electronic Institutions

J. Ll. Arcos, M. Esteva, P. Noriega, J. A. Rodríguez-Aguilar and C. Sierra

Abstract. There is an increasing need of methodologies and software tools that ease the development of applications where distributed heterogeneous entities can participate. Multi-agent systems, and electronic institutions in particular, can play a main role on the development of this type of systems. Electronic institutions define the rules of the game in agent societies, by fixing what agents are permitted and forbidden to do and under what circumstances. The goal of this paper is to introduce an integrated development environment that supports the engineering of electronic institutions.

Keywords. electronic institutions, multi-agent systems, software engineering.

1. Introduction

Multi agent systems (MAS) are systems composed of autonomous agents which interact in order to satisfy their common and/or individual goals. A main feature of MAS is that the communication occurs at knowledge level and that they use flexible and complex interactions among their components. Thus, the design and development of MAS suffer from all the problems associated to the development of distributed concurrent systems and the additional problems which arise from having flexible and complex interactions among autonomous entities [13]. The complexity of designing multi-agent systems increases when we focus on open systems [11]. Open multi agent systems are those in which the participants are unknown in advance and can change over time. These systems are populated by heterogeneous agents, generally developed by different people using different languages and architectures, representing different parties, and acting to maximise their own utility. In order to cope with these problems appropriate methodologies that allow the analysis and design of agent systems and software tools that support their development life cycle are needed [13, 12].

Human societies successfully deal with similar issues by deploying institutions [15] that establish how interactions of a certain sort will and must be structured within an organization. Institutions represent the rules of the game in a society, including any (formal or informal) form of constraint that human beings devise to shape human interaction. Therefore, they are the framework within which human interaction takes place, defining what individuals are forbidden and permitted and under what conditions. Furthermore, human institutions not only structure human interactions but also enforce individual and social behaviour by obliging everybody to act according to the norms.

Hence, we advocate for the introduction of their electronic counterpart, namely *electronic institutions* (EIs) [14, 18, 8], to establish the rules of the game in agent societies. An EI defines a set of artificial constraints that articulate agent interactions, defining what they are permitted and forbidden to do. An EI defines a normative environment where heterogeneous (human and software) agents can participate by playing different roles and can interact by means of speech acts [21]. Our actual experience in the deployment of actual-world MAS as EIs [19, 4] allow us to defend the validity of this approach. Notice though, that as noted in [8, 18] we believe that engineers need to be supported by well-founded tools. Hence, in this paper we introduce an integrated development environment for EIs that supports engineers through all the stages of the design and development of MAS.

As a case study we will use through this paper the Double Auction Market [9]. In this institution trader agents participate for selling and buying different commodities. The institution offers participating traders a brokering service in which trader agents can register which commodities they are interested on selling and buying. Therefore, when there are some trader agents interested in one commodity the institution realises a double auction to facilitate the trading.

The paper is structured as follows. In section 2 we describe the components of our EI model. From section 3 to 7 we outline the different stages for engineering EIs and how the different software tools that we have developed support them. Finally, in sections 8 and 9 we describe the related work and draw some conclusions.

2. Electronic Institutions. Fundamental Concepts

In this section we present a short description of electronic institutions (for further details refer to [14, 18, 6]). To define an electronic institution it is necessary to define a common language that allows agents to exchange information, the activities that agents may do within the institution and the consequences of their actions. Our model of electronic institutions is based on four principal elements: a dialogical framework, a set of scenes, a performative structure, and a set of normative rules.

The dialogical framework defines the valid illocutions that agents can exchange and the participant roles as well as their relationship. By sharing a dialogical framework, we enable heterogeneous agents to exchange knowledge with

other agents. 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 *dialogical framework*. EIs 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 dialogical framework defines the possible participating roles within the EI 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 [20]. In the context of an EI, we distinguish between two types of roles: *internal* and *external*. The internal roles can only be played by what we call *staff* agents, i.e. the agents that belong to the institution. We can regard them as the electronic version of the workers in human institutions. Since an institution delegates its services and duties to the internal roles, an external agent can never play an internal role.

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 agents group meetings, which we call *scenes*, that follow well-defined communication protocols. In fact, no agent interaction within an EI takes place out of the context of a scene. We consider the protocol of each scene to model the possible dialogic interactions between roles. 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, depending on their role, either to enter or to leave a scene at some particular moments(states) of an ongoing conversation.

A scene protocol is specified by a directed graph whose 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, it is defined what can be said, by whom and to whom. As we want the protocol to be generic, state transitions cannot be labelled by grounded illocutions, instead illocution 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. Thus, the protocol is independent of concrete agents and time instants. Moreover, arcs labelled with illocution schemes can have some constraints associated which impose restrictions on the valid illocutions and on the paths that the conversation can follow. This allows, for instance, to specify that in an auction scene following the English auction protocol, buyers' bids must be always greater than the last submitted bid.

While a scene models a particular multi-agent dialogic activity, more complex activities can be specified by establishing relationships among scenes which are captured in the performative structure. 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. In order to capture the relationship between scenes we use a special type of scenes: *transitions*. The type of transition allows to express agents synchronisation, choice points where agents can decide which path to follow or parallelisation points where agents are sent to more than one scene. Transitions can be regarded as a type of routers in the context of a performative structure. Moreover, the very same agent can be possibly participating in multiple scenes at the same time. Likewise, there may be multiple concurrent executions of a scene. Therefore 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.

A performative structure can be regarded as a network of scenes whose connections are mediated by transitions that determine the role flow policy among different scenes. That is, *how* agents depending on their role can move among the different scenes and *when* new conversations can be started. Finally, an initial and a final scene determine the entry and exit points of the institution respectively.

Agent actions in the context of an institution have consequences, usually in the shape of compromises which impose obligations or restrictions on dialogic actions of agents in the scenes wherein they are acting or will be acting in the future. The purpose of normative rules is to affect the behaviour of agents by imposing obligations or prohibitions.

Notice that since we are considering dialogical institutions the only actions that we consider are the utterance of illocutions. In order to refer to performed actions within normative rules we set out two predicates:

- $uttered(s, w, \mathbf{i})$ denoting that a grounded illocution unifying with the illocution scheme $\mathbf{i}$ has been uttered at state w of scene S identified by s.
- $uttered(s, \mathbf{i})$ denoting that a grounded illocution unifying with the illocution scheme $\mathbf{i}$ has been uttered at some state of scene S identified by s.

Therefore, we can refer to the utterance of an illocution within a scene or when a scene execution is at a concrete state.

Definition 2.1. Normative rules are first-order formulae of the form

$$\left(\bigwedge_{j=1}^{n} uttered(s_j, w_{k_j}, \mathbf{i}_{l_j}) \wedge \bigwedge_{k=0}^{m} e_k\right) \rightarrow \left(\bigwedge_{j=1}^{n'} uttered(s'_j, w'_{k_j}, \mathbf{i}'_{l_j}) \wedge \bigwedge_{k=0}^{m'} e'_k\right)$$

where s_j, s'_j are scene identifiers, w_{k_j}, w'_{k_j} are states of s_j and s'_j respectively, $\mathbf{i}_{l_j}, \mathbf{i}'_{l_j}$ are illocution schemes l_j of scenes s_j and s'_j respectively, and e_k, e'_k are boolean expressions over variables from the illocution schemes $\mathbf{i}_{l_j}$ and $\mathbf{i}'_{l_j}$.

The intuitive meaning of normative rules is that if grounded illocutions matching $\mathbf{i}_{l_1}, \ldots, \mathbf{i}_{l_n}$ are uttered in the corresponding scene states, and the expressions $e_1, \ldots, e_m$ are satisfied, then, grounded illocutions matching $\mathbf{i}'_{l_1}, \ldots, \mathbf{i}'_{l'_n}$ satisfying the expressions $e'_1, \ldots, e'_{m'}$ must be uttered in the corresponding scene states. Notice that $\mathbf{i}'_{l_1}, \ldots, \mathbf{i}'_{l'_n}$ can be regarded as *obligations* that agents acquire

where the antecedent of the normative rule is satisfied. Therefore, agents must utter grounded illocutions matching these illocutions schemes and satisfying $e'_1, \ldots, e'_{m'}$, in the corresponding scenes in order to fulfil the normative rule.

To summarise, the notions above picture the regulatory structure of an EI as a "workflow" (performative structure) of multi-agent protocols (scenes) along with a collection of (normative) rules that can be triggered off by agents' actions (speech acts). Notice too that the formalisation of an EI focuses on macro-level (societal) aspects, instead of on micro-level (internal) aspects of agents.

3. Engineering Electronic Institutions

Next we detail the steps to be followed when engineering and subsequently executing institutions. These steps cover the engineering of both the institutional rules and the participating agents. Thus, we propose the engineering of EIs through the following stages:

- *Specification.* Formal specification of the institutional rules. In other words, a formal specification of EIs concepts introduced in section 2. The result is a precise description of the kinds and order of messages that agents can exchange, along with a collection of norms to regulate their actions.
- *Verification.* Once specified an institution, it should go through a verification process before opening it to participating agents. This step is twofold. There is a first verification process focusing on static, structural properties of the EI specification. A second verification process follows concerned with the expected dynamic properties of the EI. We advocate that the dynamic verification of EIs should be done by means of simulation, with the aim of exploring the institution performance with different populations of agents. The institution designer should analyse the results of the simulation and eventually introduce changes in the specification if they differ from the expected ones.
- *Agent generation.* Once the institution specification is validated, it can be made available for agent participation. At this point, agent designers have to implement their agents. We want to remark that we do not impose restrictions on the type of agents that can participate in the institution. Hence, agent designers can choose the language and architecture that better fulfill their goals.
- *Execution & Analysis.* An EI defines a normative environment that shapes agent interactions. As an institution will be populated at execution time by heterogeneous and self-interested agents, we cannot expect that these agents will behave according to the institutional rules encoded in the specification. For this purpose, we advocate that the institution should be executed via an infrastructure that facilitates agent participation, while enforcing the institutional rules. Furthermore, it is important to give support to the monitoring of EIs executions to detect agents' misbehaviours and unexpected situations.

In order to facilitate the engineering of EIs we have developed a set of software tools that give support to all the above-mentioned steps. These tools are integrated in the Electronic Institutions Integrated Development Environment (EIDE). EIDE allows for engineering both the institutional rules and the participating agents. EIDE is composed of the following tools:

ISLANDER: A graphical tool that supports the specification and static verification of institutional rules.

SIMDEI: A simulation tool to animate and analyse *ISLANDER* specifications. In other words, SIMDEI supports the dynamic verification of *ISLANDER* specifications.

aBUILDER: An agent development tool that, given an *ISLANDER* specification from an EI, generates agent skeletons. The generated skeletons can be used either for EI simulations supported by *SIMDEI* or for actual executions of the institution supported by *AMELI*.

AMELI: A software platform to run EIs. The platform facilitates agent participation in the institution while enforcing the institutional conventions. Electronic institutions specified with *ISLANDER* are run by *AMELI*.

Monitoring tool: A tool that permits the monitoring of EI executions run by *AMELI*. It graphically depicts all the events occurring during an EI execution.

In what follows we describe how the different EIDE tools are employed for the engineering of EIs. In order to illustrate how the different tools work, we use the Double Auction Market.

4. Specification and Vverification via ISLANDER

The specification and static verification of EIs is supported by *ISLANDER* [7]. As to the specification of EIs, the tool combines both graphical and textual specifications. More precisely, the tool permits the graphical specification of the roles and their relationships, the scene protocols, and the performative structure. We believe that graphical specifications facilitate the work of agent designers as they are easier to create and to understand. In order to textually define the remaining elements of a specification, the tool structures the way in which it has to be introduced. Whenever possible, pop-down menus are used. This is used for fields that contain references to other specified elements and for fields whose value is one out of a predefined set. This facilitates the designer's work because he has only to select the element from a list and thus reduces typing errors.

ISLANDER supports the static verification of specifications by checking their structural correctness. Thus, the tool carries out the following verifications:

- **Integrity.** The tool checks that cross references among the different elements of the specification are correct. In other words, it checks that each element which is referenced is actually defined.

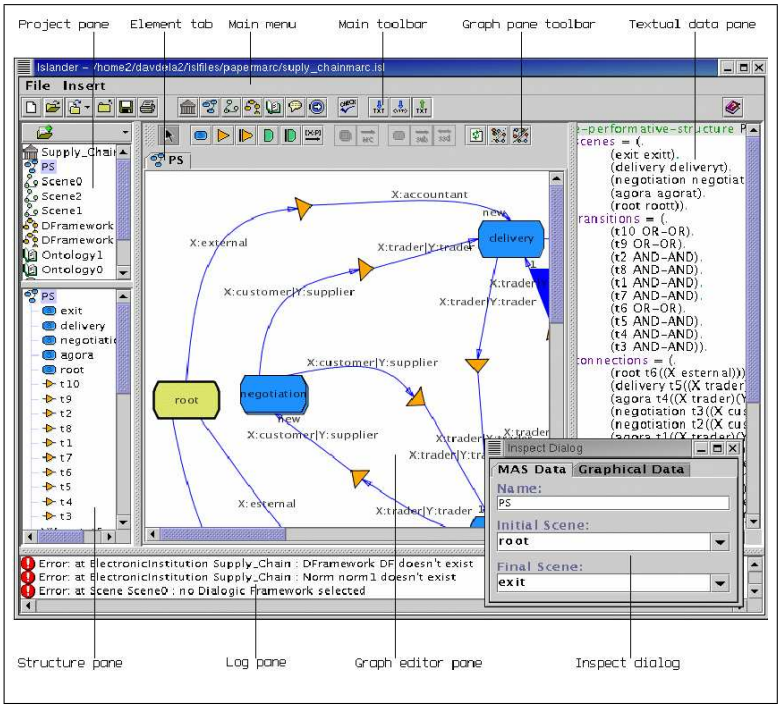

FIGURE 1. Islander GUI

- **Liveness.** It checks that agents will not be blocked at any point of the performative structure, that each scene is reachable for each of its roles, and that agents can always reach the final scene from each scene.
- **Protocol Correctness.** It checks that scene protocols are correctly specified. That is, that each state of the graph is accessible from the initial state, that a final state is reachable from each state and that the labels of the different arcs are correctly specified according to the scene dialogical framework.
- **Normative rules correctness.** It checks that normative rules are specified correctly and that agents can reach the scenes where they have to utter the illocutions for fulfilling each normative rule.

Figure 1 depicts the graphical user interface of *ISLANDER*. The menus contain the general operations of the application and they are similar to other applications. For instance, the file menu contains options to save the current specification, and open a new one, while the insert menu pemits the user to insert a new EI component on the current specification. The tool bar contains icons for a quick access to the operations. There is also a specific icon that activates the verification process. On the *project structure pane* the user can see all the elements

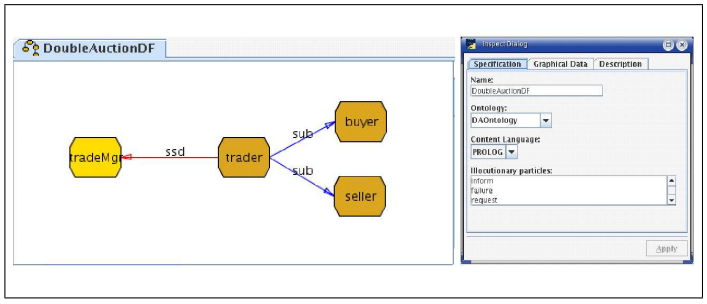

FIGURE 2. Double Auction Dialogical Framework

and sub-elements that belong to the current specification ordered by category. It permits the user to navigate among them. When he changes the selection the other panes are modified appropriately in order to show the information of the selected element or sub-element. The *graph editor pane* supports the edition of the graphical components of EIs, namely the edition of the graphical component of performative structures, dialogical frameworks and scenes. The user can edit the different graphs and modify them using the mouse. The graph editor pane is used for the creation and modification of the graph topology while the textual information associated to the graph is introduced and modified using the *inspect pane*. For instance, the graph pane is used for adding a new node to the graph but the name of the node is introduced using the inspect pane. Furthermore, the inspect pane is used for the specification of the rest of the elements of an EI. Finally, the *verification message pane* at the bottom of the figure, is used for showing the errors when the user activates the verification of the current specification. Moreover, *ISLANDER* allows the user to move to the element containing the error by simply selecting the error message text. When a user selects an error all the panes of the application are modified in order to show the element containing the error. Once an EI has been specified and verified, the user can export the institution to XML, employed at further stages of the development cycle.

Summarising, *ISLANDER* supports the specification of EIs, facilitating the designer's work by combining graphical and textual specifications. Furthermore, the verification process permits checking the correctness of the specifications. The result of this stage is a sound, unambiguous and correct specification of the institutional rules. In the next subsections we present the specification of the double auction market using *ISLANDER*.

4.1. Specifying the Double Auction Market

4.1.1. Dialogical Framework. Figure 2 depicts the specification of the Double Auction Market dialogical framework. Notice that there are four different roles, namely: $tradeMgr$, $trader$, $buyer$ and $seller$. They are specified graphically using the graph

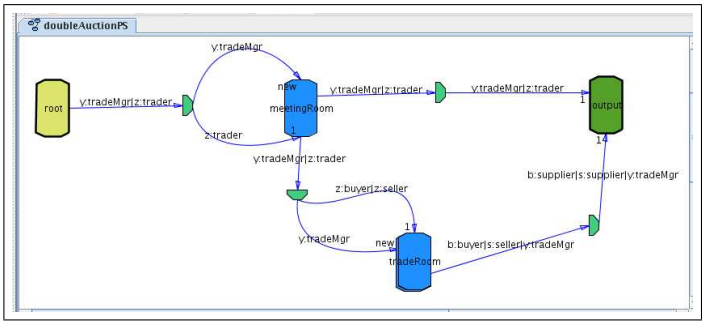

FIGURE 3. Double Auction Performative Structure

editor pane. In order to differentiate between the internal and external roles, internal roles are displayed in yellow, while external roles are displayed in brown. Hence, we can see that there is only one internal role ($tradeMgr$), whereas there are three external roles (*trader*, *buyer* and *seller*). The $tradeMgr$ role will be played by a staff agent in charge of the brokering service and in charge of starting and realising the double auctions. The rest of the roles will be played by the external agents entering the institution for buying and selling commodities.

Furthermore, there are also some relationships among roles. On the one hand, the $tradeMgr$ and the *trader* roles are incompatible (denoted by the *ssd* relation in figure 2) meaning that agents cannot play both roles within the institution. On the other hand, the *trader* role subsumes (denoted by the *sub* relation in figure 2) the *buyer* and *seller* roles, meaning that agents authorised to play the trader role can also play the buyer and seller roles.

Finally, the specification contains the definition of the rest of the elements of a dialogical framework, namely: the ontology, the content language and the illocutionary particles. Concretely, the specification defines that the institution ontology is the *DAOntology*, that the content language is PROLOG and that the illocutionary particles are *inform*, *failure* and *request*.

4.1.2. Performative structure. Figure 3 shows the specification of the performative structure of the Double auction market as shown by *ISLANDER*. Its activities are represented by the *meetingRoom* scene, where traders are matched by the trade manager based on their interests in commodities, and the *tradeRoom* scene, where a Double auction is run to rule the trading. Observe that trading agents switch their role to either buyer or seller when moving from the *meetingRoom* to the *tradeRoom*. Notice too that while there is a sole execution of the *meetingRoom* scene, multiple executions of the *tradeRoom* scene may occur, being dynamically created depending on trading agents' interests. Finally, the *root* scene and the *output* scene represent the institution's entry and exit points.

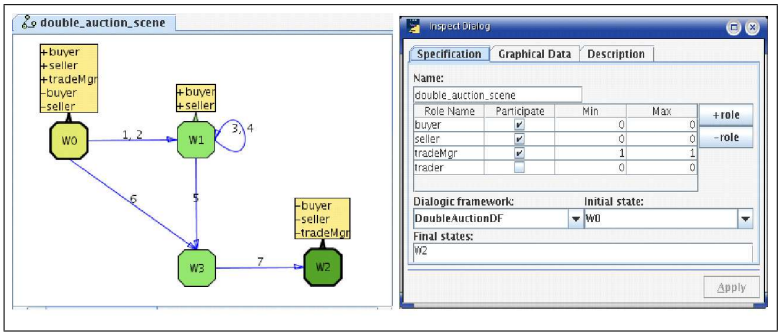

FIGURE 4. Double Auction scene

4.1.3. Double Auction Scene. Figure 4 depicts the specification of the double auction scene where commodities are traded following a double auction protocol. In this protocol agents playing the buyer and seller roles have some time to submit their offers, which are matched later on to decide which transactions are done.

The first issue to address when specifying a scene is the definition of the roles that can participate in it and size of their populations. In this scene can participate agents playing the $tradeMgr$, *buyer* and *seller* roles. The scene requires the participation of exactly one agent playing the $tradeMgr$ role, while there are no constraints on the number of agents playing the *buyer* and *seller* roles (zero is the default value for the *Max* field for a role meaning that there is no limit on the number of agents that can participate playing the role). We can also observe that the dialogical framework, that will be used to construct the illocution schemes labeling the arcs of the scene protocol, is the *DoubleAuctionDF*.

On the right part of the figure we can observe the specification of the scene protocol introduced using the graph editor panel. The following labels are associated to the arcs:

1 *inform(?s:seller, ?t:tradeMgr, offer(?offer))*
2 *inform(?b:buyer, ?t:tradeMgr, demand(?demand))*
3 *inform(?s:seller, ?t:tradeMgr, offer(?offer))*
4 *inform(?b:buyer, ?t:tradeMgr, demand(?demand))*
5 *[5000]*
6 *[5000]*
7 *inform(?t:tradeMgr,all,performed_contracts(?contract))*

The scene starts at its initial state $w0$ where buyers and sellers can start to submit their offers, labels 1 and 2. Notice that after the first offer is made the scene will evolve to state $w1$, where they can continue submitting offers, labels 3 and 4, until the timeout specified by label 5 expires making the scene evolve to $w3$. Furthermore, the connection from $w0$ to $w3$ labeled also with a timeout guarantees that the scene will not be blocked if there are no offers. Once at state $w3$ the $tradeMgr$ calculates the contracts matching buyers and sellers offers using

the rules of the double auction protocol. The scene concludes when the $tradeMgr$ announces the contracts, label 7, making the scene evolve to its final state $w2$.

Finally, in the figure it can be observed that buyer and seller agents can enter the scene at states $w0$ and $w1$, while they can leave the scene at the states $w0$ and $w2$. Complementarily, the $tradeMgr$ agents can enter the scene at $w0$ and can leave it at $w2$.

5. Dynamic verification via SIMDEI

While *ISLANDER* permits the static verification of EIs, their dynamic verification is done via simulation. The purpose of the simulation is to study the dynamic behaviour and performance of the specified institution under different circumstances. For instance, in the case of the double auction the institution designer could simulate the performance of the system with different populations of buyer and seller agents.

Simulations of EIs can be run by means of the *SIMDEI* simulation tool that we have developed over REPAST [1]. Before running the simulation institution designers must develop different type of agents playing the different institution roles. This process is partially supported by the *aBUILDER* tool as explained in the next section. It is a task of the institution designers to develop types of agents, as similar as possible to the ones that will populate the institution when it will be open to external agents. Once agents have been developed simulations can be executed thanks to the *SIMDEI* simulation tool to conduct *what-if analysis*. The institution designer is in charge of analyzing the results of the simulations and return to the specification stage if they differ from the expected ones.

6. Agent development via aBUILDER

Once the institution has been specified and verified, it is time for designing the agents. Notice that within an institution we distinguish between the internal roles played by the staff agents and the external roles played by the external agents. Since staff agents are those in which the institution delegates their services and duties, they are necessary for the correct execution of the institution. Thus, it is a task of institution designers to develop them. On the contrary, external agents playing the external roles must be developed by the parties that they will represent. At this point we want to remark that we do not impose restrictions on the type of agents which can participate in the institution. Agent designers can choose the language and architecture that is better to fulfill their goals as well as they can use any software tools that facilitate their work. Therefore, it is not mandatory for them to use the *aBUILDER* tool.

Agent development is partially supported by the *aBUILDER* tool. Notice that an EI specification defines what agents can do within the institution but it

[1] http://repast.sourceforge.net

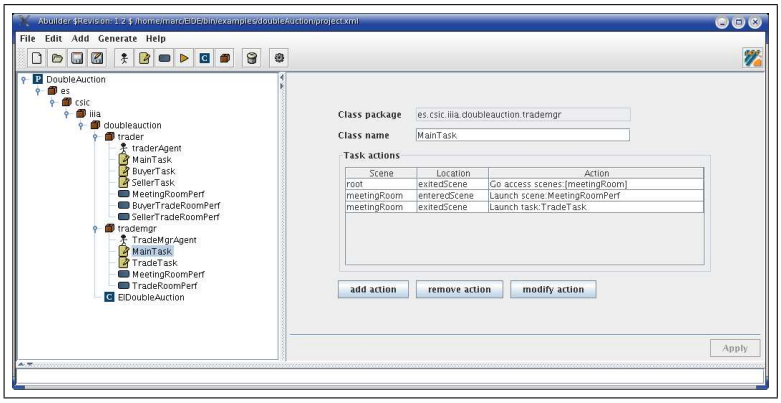

FIGURE 5. *aBUILDER* GUI

does not define how agents must take their decisions. For instance, the specification of the double auction market defines how and when buyer and seller agents can submit their offers on the double auction scene, but it is a decision of every agent to decide which offers to submit. Of course, this will depend on each agent preferences and on their decision making mechanisms, which are probably different for each agent. Given an EI specification *aBUILDER* allows for defining agent skeletons that must be later on completed by agent designers with the decision making mechanisms.

The current version of the *aBUILDER* tool permits to define agents composed by tasks and behaviours. On the one hand, tasks define general activities for an agent and each one can involve the participation in different scenes. On the other hand, a behaviour defines what an agent will do within a scene. In other words, a behaviour defines which information an agent stores from the received messages, and which utterances it utters. Therefore, the tasks determine how agents will be moving among the performative structure scenes, while the behaviours define what agents will do within the different scenes.

In order to specify a task an agent designer has to define what the agent has to do when leaving and entering scenes. On the one hand, it has to be specified which scene to join next or which task to launch when the agent leaves a scene. On the other hand, it has to be specified which behaviour to launch when the agent enters the scene. In order to know the different scenes and the rest of the institutional rules, the *aBUILDER* tool is capable of loading EIs specification as generated by *ISLANDER*.

Figure 5 depicts the graphic user interface of the *aBUILDER* tool to design agents for the Double Auction market. We can see on the left part the agents being defined. Notice that the tool permits the definition of multiple agents at the same time. In this case, two types of agents are defined, namely: $traderAgent$ and $tradeMgrAgent$. We can also observe the tasks and behaviours defined for each of

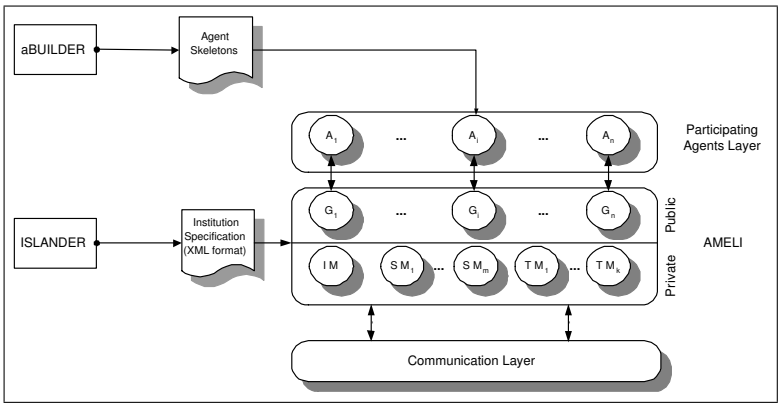

FIGURE 6. Electronic institution architecture

them. For instance, for the tradeMgr two tasks ($MainTask$ and $TradeTask$), and two behaviours ($MeetingRoomPerf$ and $TradeRoomPerf$) are defined. When a user selects any of them, its information is shown and can be modified on the right part of the *aBUILDER* graphic user interface. Concretely, in figure 5 the information shown on the right part correspond to the $MainTask$ of the $tradeMgrAgent$, which is the task that will be executed when it will enter in the institution. It can be observed that three actions are defined within this task. The first one defines that the agent will go to the *meetingRoom* scene when leaving the *root* scene. The second one defines that the $MeetingRoomPerf$ behaviour will be launched when entering the $MeetingRoom$ scene. Finally, the third one specify that the $TradeTask$ task will be launched when leaving the $MeetingRoom$ scene.

Once all the tasks and behaviours have been specified the tool permits the generation of code standing for agent skeletons for the different types of agents. In the current version the code is generated in JAVA. The generated code is capable to navigate through the institution performative structure and scenes, launching the specified tasks and behaviours at the defined points. However, they must be filled up with the decision making mechanisms before to send them to the institution.

Summarising, given an EI specification the *aBUILDER* tool supports the definition of different types of agents and the generation of agent skeletons.

7. Execution and Analysis

7.1. Execution via AMELI

As depicted in figure 6, our architecture is composed of the following layers:

- **Participating agent layer.** The agents taking part in the institution.
- **Social layer (AMELI).** An infrastructure that mediates and facilitates agents' interactions while enforcing the institutional rules.

- **Communication layer.** In charge of providing a reliable and orderly transport service. The current implementation can either use JADE [1] or a publish-subscribe event model as communication layer.

Notice that unlike approaches that allow agents to openly interact with their peers via a communication layer, we advocate for the introduction of a social layer (*AMELI*) which mediates agent interactions at run time. On the one hand, *AMELI*provides participating agents with information about the current execution. For instance, information about the participating agents within a scene execution. On the other hand, it enforces the institutional rules to the participating agents. At this aim, *AMELI* keeps track of the execution state, and uses it along with the institutional rules encoded in the specification to validate agents actions.

As we can observe in figure 6 *AMELI* is composed of the following types of agents:

- **Institution Manager (IM)**. It is in charge of starting an EI, authorising agents to enter the institution, as well as managing the creation of new scene executions. It keeps information about all participants and all scene executions. There is one institution manager per institution execution.
- **Transition Manager (TM)**. It is in charge of managing a transition controlling agents' movements towards scenes. There is one transition manager per transition.
- **Scene manager (SM)**. Responsible for governing a scene execution (one scene manager per scene execution).
- **Governor (G)**. Each one is devoted to mediating the participation of an agent within the institution. There is one governor per participating agent.

Since external agents can only communicate with their governors, we can regard *AMELI* as composed of two layers: a *public* layer, formed solely by governors; and a *private* layer, formed by the rest of *AMELI* agents, not directly accessible to external agents. Furthermore, all these agents collaborate to guarantee the correct evolution of an EI execuution.

Finally we want to outline the main features of our architecture:

- **Domain independent** *AMELI* can be used for the deployment of any specified institution without any extra coding. For this purpose, agents composing *AMELI* load institution specifications as XML documents generated by *ISLANDER*. Thus, the implementation impact of introducing institutional changes amounts to the loading of a new (XML-encoded) specification.
- **Agent-architecture neutral** Participating agents are only required to be capable of opening a communication channel with their governors.
- **Scalable** When employing JADE, the agents composing *AMELI* can be readily distributed among different machines
- **Communication neutral** Participating agents regard our architecture as *communication neutral* since they are not affected by changes in the communication layer.

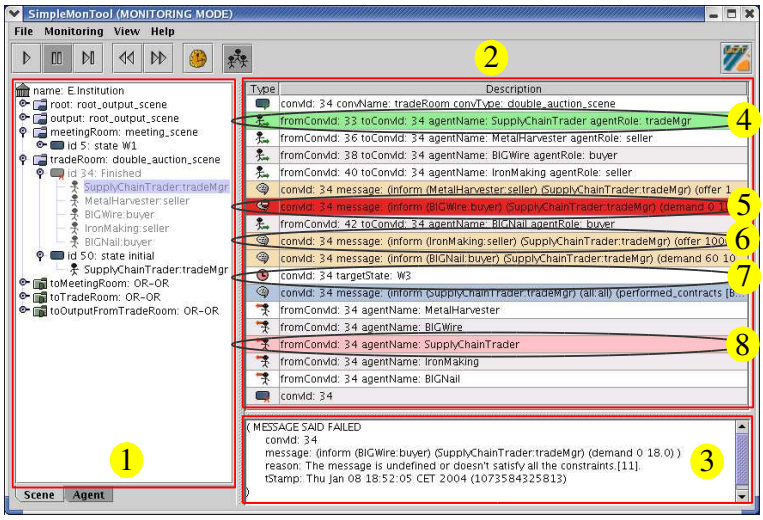

FIGURE 7. Double auction market monitoring

The execution of an EI starts with the creation of an Institution Manager. Once up, the institution manager activates the initial and final scenes launching a scene manager for them. Thereafter, external agents can begin submitting to the institution manager their requests to join the institution. When an agent is authorised to join the institution, it is connected to a governor and admitted into the initial scene. From there on, agents can move around the different scene executions or start new ones according to the EI specification and the current execution state.

7.2. Execution Monitoring

An EI execution can be monitored thanks to the *monitoring tool* that depicts graphically all the events occurring at run time. Fairness, trust and accountability are the main motivations for the development of a monitoring tool that registers all interactions in a given enactment of an electronic institution [14, 18]. Giving accountability information to the participants increases their trust in the institution. This is specially important for electronic institutions where people delegate their tasks to agents. Furthermore, the tool permits them to analyse their agent(s) behaviour within the institution in order to improve it. From the point of view of the institution designers, the tool is useful for testing the system and the staff agents before making the institution available to external agents. Furthermore, when the institution is running it can be used to detect unexpected situations and fraudulent behaviours of external agents.

The monitoring tool displays all the interactions occurring in the different scene and transition executions, along with agents' movements among scenes. Figure 7 shows the monitoring of an execution of the Double Auction market. Frame 1 contains a list of the institution's scenes and transitions along with their executions. The list includes a single execution of the *meetingRoom* scene (*id 5*) at state *W1*. Furthermore, there are two different executions of the *tradeRoom* scene: one ongoing execution (id 50 at the initial state), and a finished one (id 34). The figure shows that while five agents (a trade manager –*tradeMgr*, two buyers, and two sellers) have participated in scene execution 34, a single agent (a trade manager) is waiting for buyers and sellers to join in scene execution 50. According to section 7.1, there is a scene manager agent per ongoing scene execution (e.g. id 5, 50). Besides, no scene manager agent is required any longer for scene execution 34 since it is finished. Furthermore, there is one transition manager agent per transition. Frame 2 depicts the events occurring during scene execution 34: agents' entrance (e.g. label 4), the utterance of valid (e.g. label 6) and wrong (e.g. label 5) illocutions, transitions caused by timeouts (e.g. label 7), agents' exit (e.g. label 8). We must remind the reader that the coordinated activity of the scene manager of the scene execution and the participating agents' governors guarantee that all these events abide by the scene specification. To illustrate the control of *AMELI* agents, frame 3 visualises an illocution rejected because a constraint in the specification is violated when buyer *BIGWire* attempts at submitting a demand of 0 units at 18 EUR. Since the scene manager evaluates the illocution as not valid, *BIGWire* is informed by its governor about the failed action.

Finally we wan to remark that the tool also allows for the monitoring of the participation of an agent within the institution. In this case, it shows the scenes in which the agent has taken part, along with the messages that it has exchanged.

8. Related Work

Recently, a number of MAS methodologies —e.g. GAIA [24], Tropos [10], or [22] to name a few— have been proposed. Most MAS methodologies are based on strong agent-oriented foundations, however, while offering original contributions at the design level, they tend to be unsatisfactory on a development level because of the lack of support to their design and implementation. Furthermore, most MAS methodologies are agent-centered rather than community or socially-centered, hence focus more on the internal aspects of agent functionality than on the interaction aspects.

There are some agent infrastructures such as DARPA COABS [3] and FIPA compliant platforms such as JADE [1] that deal with many issues that are essential for open agent interactions —communication, identification, synchronization, matchmaking— that can be used as building blocks for the development of open multi-agent systems. These building blocks are arguably too distant from organisation-centered patterns or social structures.

A different —and interesting— approach to a unified MAS development framework are the protocol-centered approaches. The proposal by Hanachi [2] allows for specifications of interaction protocols that need to be subsequently compiled into a sort of executable protocol brokers called moderators. In Tropos, the specifications are transformed into agent skeletons that must be extended with code, similar to the *aBUILDER* tool presented here. However, at execution time there is no mechanism to ensure that agents follow the specification of the system.

Although some proposals agree on the need of adopting a social stance, as far as we can tell the formal definition of organization-centered patterns and social structures in general, along with their computational realization, remain largely undeveloped (as noted in [24]).

In addition to these infrastructures and methodologies just mentioned, some agent research has focused on the introduction of social concepts such as organizations or institutions (e.g. [17],[5], [23]), however, there are yet no tools supporting their computational realization, nor a proper engineering methodology directly associated with them.

A promising line of work is the one adopted by Omicini and Castelfranchi (e.g. [16]). It postulates some significant similarities with our EI approach: focus on the social aspects of the interactions, a unified metaphor that prevails along the development cycle, and the construction of tools to implement methodological ideas. They discuss *coordination artifacts* and propose to develop them as devices to wrap agents so that their interactions in a given MAS are subject to that MAS protocol and keep an accurate picture of the interaction context[2]. While their proposal mentions other conceptual design levels —and, consequently, other devices— the actual development of the methodology and the associated tools appears to be still rather tentative.

9. Conclusions

We argue that open multi-agent systems, populated by heterogenous and self interested agents, can be effectively designed and developed as electronic institutions. Similarly, to human institutions in human societies, EIs define the rules of the game in agent societies, establishing what agents are permitted and forbidden to do. In other word, an EI structures the valid interactions that agents may have as well as defining the consequences of those interactions. Therefore, an EI defines a normative environment that constraints agents interactions at run time.

In order to cope with the complexity of engineering EIs, we early identified the importance of developing software tools which give support to their design, development and subsequent execution. Therefore, through this paper we have presented an Electronic Institutions Integrated Development Environment (EIDE) that supports the engineering of EIs. Through the paper we have presented the

[2]These coordination artifacts are essentially what we call *governors*.

different tools that compose EIDE, and we have illustrated how they work using as an example the Double Auction Market. Notice, that we advocate that EIs engineering must start with a formal specification of the institutional rules supported by *ISLANDER*. The result is a sound and unambiguos definition of the institutional rules. Furthermore, *ISLANDER* also supports the static verification of specifications, while dynamic verification is done via simulation using the *SIMDEI* tool. Although, we do not impose constraints on the type of agents that can partcipate in an EI, their design and development is partially suppported by the *aBUILDER* tool. Finally, EIs can be executed thanks to *AMELI*, which facilitates agent participation within an EI, while enforcing the institutional rules. A main feature of *AMELI* is that it can be used for the execution of any specified instituion without any extra coding. Furthermore, EI executions can be monitored using the monitoring tool. To conclude, we want to point out that EIDE has proven to be highly valuable in the development of actual-world e-commerce applications such as the Multi-Agent System for Fish Trading (MASFIT) [4].

For further information and software downloads, the interested reader should refer to http://e-institutions.iiia.csic.es.

10. Acknowledgements

Marc Esteva enjoys a Fulbright/MECD postdoctoral scholarship FU2003-0569. The research reported in this paper is partially supported by the Spanish CICYT project Web-i (2) (TIC-2003-08763-C02-01). The authors would like to thank the IIIA Technological Development Unit's programmers for their valuable contribution to the development of EIDE.

References

[1] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with jade. In C. Castelfranchi and Y. Lesperance, editors, *Intelligent Agents VII*, number 1571 in Lecture Notes in Artificial Intelligence, pages 89–103. Springer-Verlag, 2001.

[2] C. Sibertin-Blanc C. Hanachi. Protocol moderators as active middle-agents in multi-agent systems. *Journal of Autonomous Agents and Multiagent Systems*, 8(2), March 2004.

[3] Control of agent-based systems. http://coabs.globalinfotek.com.

[4] Guifré Cuní, Marc Esteva, Pere Garcia, Eloi Puertas, Carles Sierra, and Teresa Solchaga. Masfit: Multi-agent systems for fish trading. In *16th European Conference on Artificial Intelligence (ECAI 2004)*, Valencia, Spain, August 2004.

[5] V. Dignum. *A Model for Organizational Interaction*. PhD thesis, Dutch Research School for Information and Knowledge Systems, 2004. ISBN 90-393-3568-0.

[6] M. Esteva. *Electronic Institutions: from specification to development*. PhD thesis, Universitat Politècnica de Catalunya (UPC), 2003. IIIA monography Vol. 19.

[7] M. Esteva, D. de la Cruz, and C. Sierra. Islander: an electronic institutions editor. In *Proceedings of AAMAS 2002*, pages 1045–1052, 2002.

[8] Marc Esteva. *Electronic Institutions: from specification to development*. IIIA PhD Monography. Vol. 19, 2003.

[9] Daniel Friedman and John Rust, editors. *The Double Auction Market: Institutions, Theories, and Evidence*. Addison-Wesley Publishing Company, 1991. Proceedings of the Workshop on Double Auction Markets held June, 1991, Santa Fe, New Mexico.

[10] F. Giunchiglia, J. Mylopoulos, and A. Perini. The tropos software development methodology: Processes. Technical Report 0111-20, ITC-IRST, November 2001.

[11] C. Hewitt. Offices are open systems. *ACM Transactions of Office Automation Systems*, 4(3):271–287, 1986.

[12] Carlos A. Iglesias, M. Garijo, and J. C. Gonzalez. A survey of agent-oriented methodologies. In J. P. Muller, M. Singh, and A. S. Rao, editors, *Intelligent Agents V*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1999.

[13] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-agent Systems*, 1:275–306, 1998.

[14] Pablo Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor*. Number 8 in IIIA Phd Monograph. 1997.

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

[16] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Cristiano Castelfranchi, and Luca Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In *Third International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS'04)*, pages 286–293, New York, USA, July 19-23 2004.

[17] H. Parunak and J. Odell. Representing social structures in uml. In *Agent-Oriented Software Engineering II. LNCS 2222*, pages 1–16. Springer-verlag edition, 2002.

[18] 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.

[19] Juan A. Rodríguez-Aguilar, Pablo Noriega, Carles Sierra, and Julian Padget. Fm96.5 a java-based electronic auction house. In *Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology(PAAM'97)*, pages 207–224, 1997.

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

[21] J. R. Searle. *Speech acts*. Cambridge U.P., 1969.

[22] A. Sturm, D. Dori, and O. Shehory. Single-model mehtod for specifying multi-agent systems. In *Proceedings of AAMAS 03*, pages 121–128, Melbourne, Australia, 2003.

[23] J. Vazquez and F. Dignum. Modelling electronic organizations. In *Multi-Agent Systems and Applications III*, volume 2691 of *LNAI*, pages 584–593. Springer-verlag edition, 2003.

[24] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.

J. Ll. Arcos
Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC
08193 Bellaterra, Barcelona, Spain.
Voice: +34 93 580 95 70 Fax: +34 580 96 61
e-mail: arcos@iiia.csic.es

M. Esteva
(uiuc) Graduate School of Library and Information Science
University of Illinois at Urbana-Champaign
501 E. Daniel Street, Champaign, IL 61820
Voice: +1 217 265 0235 Fax: +1 217 244 3302
e-mail: esteva@uiuc.edu

P. Noriega
Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC
08193 Bellaterra, Barcelona, Spain.
Voice: +34 93 580 95 70 Fax: +34 580 96 61
e-mail: pablo@iiia.csic.es

J. A. Rodríguez-Aguilar
Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC
08193 Bellaterra, Barcelona, Spain.
Voice: +34 93 580 95 70 Fax: +34 580 96 61
e-mail: jar@iiia.csic.es

C. Sierra
Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC
08193 Bellaterra, Barcelona, Spain.
Voice: +34 93 580 95 70 Fax: +34 580 96 61
e-mail: sierra@iiia.csic.es