Engineering Open Environments with Electronic Institutions

Josep Ll. Arcos [a] Marc Esteva [b] Pablo Noriega [a]

Juan A. Rodríguez-Aguilar [a] Carles Sierra [a]

[a] *Artificial Intelligence Research Institute, IIIA*

Spanish Council for Scientific Research, CSIC

08193 Bellaterra, Barcelona, Spain.

{sierra,jar,pablo,arcos}@iiia.csic.es

Voice: +34 93 580 95 70 Fax: +34 580 96 61

[b] *Graduate School of Library and Information Science*

University of Illinois at Urbana-Champaign

501 E. Daniel Street, Champaign, IL 61820

esteva@uiuc.edu

Voice: +1 217 265 0235 Fax: +1 217 244 3302

Abstract

Nowadays, with the expansion of Internet, there is a need of methodologies and software tools to ease the development of applications where distributed homogeneous entities can participate. Multiagent 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 present EIDE, an integrated development environment for supporting the engineering of multiagent systems as electronic institutions.

Key words: Multiagent system, Agent Oriented Software Engineering

1 Introduction

The promises and functionality that the proposals of Open Systems anticipated in the eighties (e.g. Hewitt (Hewitt, 1986)) are now ever more pertinent for system development given the pervasiveness of IT and the added accessibility brought about by the World Wide Web. However, the challenges of building open systems are still considerable, not only because of the inherent complexity involved in having adequate interoperation of heterogeneous, independent, distributed, autonomous components, but also because of the significant difficulties of deployment and adoption of the amalgamated systems.

We have been developing a technology to address these challenges.

We do not claim to be dealing with Open Systems in their full complexity, but rather addressing a restricted —albeit significant enough— type of openness: that present in interactions that involve autonomous, independent entities that are willing to conform to a common, explicit, set of interaction conventions. We will call these *a-open systems* [1] .

For that type of open systems we have been engineering an artifact that focuses in the interactions and their compliance. We call it *Electronic Institutions* (EIs).

[1] Openness is limited by the *adscription* to the conventions

The idea behind EIs is to mirror the roles traditional institutions play in the establishment of "the rules of the game"–a set of conventions that articulate agents' interactions– but in our case applied to agents (humans or software entities) that interact through messages whose (socially relevant) effects are known to interacting parties. The essential roles EIs play are both descriptive and prescriptive: the institution makes the conventions explicit to participants, and it warrants their compliance [2].

EIs —as artifacts— involve a conceptual framework to describe agent interactions as well as an engineering framework to specify and deploy actual interaction environments. In this paper we look into the EI artifact from a methodological perspective: we discuss the notions that underly the conceptual framework and show how the system development process can be carried out with the ad-hoc software tools we have developed. We have been developing the EI artifact for some time and advocating that open MAS can be properly designed and implemented with it, as witnessed by some of the group's publications (Noriega, 1997; Esteva et al., 2001; Rodríguez-Aguilar, 2001; Esteva, 2003). Our experiences in the deployment of applications as EIs, e.g. (Rodríguez-Aguilar et al., 1997; Cuní et al., 2004) make us confident of the validity of this approach.

In what follows, in fact, we will look into EIs as a framework for developing multiagent systems (MAS). We do so for two reasons, first because a-open systems can be viewed as a type of MAS, where the entities that interoperate in the open system are simply thought of as agents. Secondly, because, in that

[2] In terms of Simon's engineering design abstractions, EIs are the –social– interface layer between the problem space the participating systems deal with, on one side, and the internal decision or functional intricacies of the various participating systems, on the other.

light, some recent methodologies and conceptual proposals for MAS engineering are then relevant for a-open systems. Our approach, as we shall show, has things in common with some of those methodologies and conceptual proposals, however we believe that it contributes to the engineering of this type of MAS through three salient distinctive features:

(1) It is socially-centered, and neutral with respect to the participating agents internals and the application domain of their interactions.
(2) It has a uniform conceptual framework to manage components and interactions that prevails through the different views (high-level specification, implementation, monitoring, ...) of a given system.
(3) It has an interaction-centered methodology that is embedded in a suit of software tools that support the system development cycle from specification to deployment.

In this paper we will illustrate how the EI framework can be used to engineer full-fledged a-open MAS. In Section 2 we discuss the ideas that constitute the conceptual EI framework and in Section 3 the tools we have developed to operationalize our methodology. In Section 4 we sketch an actual EI-based system that we developed using the EI artifact.

2 Electronic Institutions

We mentioned, following D.C. North (North, 1990), that traditional institutions can be viewed as "a set of artificial restrictions that articulate agent interactions". Analogously, when looking at computer-mediated interactions we think of Electronic Institutions as a regulated virtual environment where

the relevant interactions among participating entities take place.

This crude picture may become sharper by describing the theoretical components that operationalize it. We start by making some operational assumptions explicit:

(1) Participating entities are agents. In the accepted sense of being persistent, identifiable, communication-capable software or humans, capable of adopting commitments.
(2) Interactions are repetitive.
(3) All interactions are speech acts. That is, any and every interaction is –or is tagged by– a message that has some effect on the shared environment where agents interact.
(4) Only illocutions uttered by participating agents have effect on the shared environment.

All things considered these are rather innocent assumptions whose basic purpose is to facilitate the definition of a regulated environment. Assumption 1 is simply a convenient use of terminology that turns EIs into a sort of MAS without loss of content either way. Assumptions 2, 3, and 4, is what we have called the "dialogical stance" by which we conceive interactions as repetitive dialogues. This dialogical stance is mostly a conventional device that brings dialogical notions –and performatives– into our framework, it allows for a convenient intuitive descriptions of many EI features such as *scenes* and *performative structure* but it burdens other –like *scene transitions*– with some artificiality. Assumptions 3 and 4 are needed to operationalize the normative character of the interaction environment.

We may now get into clarifying what we mean by "relevant interactions in

a regulated environment". In order to do that we will discuss the three constituent elements of our theory for electronic institutions. Firstly, the *Dialogical Framework* that allows us to express the syntactic aspects of EIs, and the ontology of a particular EI. Then the two other elements that allow us to express the prescriptive aspects of EIs and, in particular, what the social effects of the speech acts are intended to be.

2.1 Dialogical Framework

A traditional institution, say an auction house, restricts and gives meaning to interactions participants engage in, and sees to it that the consecuences of any interaction that takes place within the institution actually happen. In an auction house, for example, if a good is being offered, the only action buyers can take is to rise their hand, indicating they take the bid; any other action is meaningless or inadmissible (and interpreted as a silent "no" to the bid). If a buyer wins a bid, the auctioneer will adjudicate the good to the buyer, charge the buyer and pay the seller for it; thus making the interactions involved relevant and meaningful to all participants. If we want to be able to build an electronic auction house we should be able to express that kind of conventions in a way that can be implemented, adscribed to by independent agents, and –basic to our purpose– in such a way that actual transactions can properly be carried out in the implemented institution.

We need to settle on a common illocutory language that serves to tag all pertinent interactions, or more properly, the valid speech acts.

Formally, we take interactions to be illocutory formulas:

$\iota(speaker, hearer, \phi, t)$

Speech acts that we use start with an illocutory particle (declare, request, promise) that a *speaker* addresses to a *hearer*, at a time t, and the content ϕ of the illocution is expressed in some object language whose vocabulary is the EI's ontology. [3]

To fill in these formulas we need vocabulary and grammar, but we also need to refer to speakers and listeners, actions, time. We call all that the *Dialogical Framework* because it includes all what is needed for agents to participate in admissible dialogues in a given EI.

To make clear what are all the available illocutions for agent dialogues in a given institution we define a *dialogical framework* as a tuple:

$DF = \langle O, L, I, R_I, R_E, R_S \rangle$

where

(1) O stands for the EI domain ontology;
(2) L stands for a content language to express the information exchanged between agents;
(3) I is the set of illocutionary particles;
(4) R_I is the set of internal roles;
(5) R_E is the set of external roles;
(6) R_S is the set of relationships over roles.

Recall that, like in a real auction house, a given agent may act as a buyer at

[3] We take a strong nominalistic view, the institutional ontology is made of every entity referred to in any admissible speech act or in any of the norms (conventions) that govern those acts and their consequences.

some point and as a seller at another, and many agents may act as buyers. This consideration allows us to think of participants adopting *roles*. Roles allow us to abstract from the individuals, the specific agents, that get involved in an institution's activities. Hence, each agent participating in an EI is required to adopt some role. All agents adopting a given role should be guaranteed to have the same rights, duties and opportunities. We differentiate between the internal and the external roles. The internal roles define a set of roles that will be played by *staff* agents which correspond to employees in traditional institutions. Since an EI delegates their services and duties to the internal roles, never an external agent is allowed to play any of them. Finally, we found useful to define relationships among roles, for instance, roles that cannot be played at the very same time, or roles that have some authority over others.

Through this DF we have all utterances that may ever be admissible in a given EI, we now turn to the task of characterising admissible dialogues.

2.1.1 Performative Structure: Scenes and Transitions

We assumed repetitive interactions are typical of institutions. For instance, in an auction house every new good is auctioned out in the same way that the previous one was offered, and the previous,... Bidding rounds are repeated dialogues very much like the scenes in a play: actors playing given roles (buyers, auctioneer) repeat their lines every time they get into the same *scene*. But in theater as in our auction house example, interactions are usually rather involved. In our example of an auction house: goods are introduced, goods are sold, goods are payed for, sellers get their money,... As in a play, some scenes precede others, some scenes get repeated; the play has a *performative*

structure.

Likewise, activities in an electronic institution are organised in a *performative structure* as the composition of multiple, distinct, and possibly concurrent, dialogical activities, each one involving different groups of agents playing different roles. Interactions between agents are articulated through recurrent dialogues which we call *scenes*, each scene following some type of conversation protocol. The protocol of each scene restricts the possible dialogical interactions between roles. Scenes represent the context in which the uttered illocutions must be interpreted. Consequently, the same message in different contexts may certainly have different meanings. A distinguishing feature of scenes is that their participants may change, as agents are allowed either to enter or to leave a scene at some particular moments (states) depending on their role.

A scene protocol is specified by a directed graph whose nodes represent the different states of a dialogical interaction between roles. Its arcs are labelled with illocution schemata from the scene's dialogical framework (whose sender, receiver and content may contain variables) or timeouts.

More formally, a *Scene* is a tuple:

$s = \langle R, DF_S, W, w_0, W_f, (WA_r)_{r \in R}, (WE_r)_{r \in R}, \Theta, \lambda, min, Max \rangle$, where

(1) R is the set of scene roles involved in that scene;
(2) DF_S is the restriction to the scene s of the EI dialogical framework as defined above;
(3) W is the set of scene states;
(4) $w_0 \in W$ is the initial state;
(5) $W_f \subseteq W$ is the set of final states;

(6) $(WA_r)_{r \in R} \subseteq W$ is a family of sets such that WA_r stands for the set of access states for role $r \in R$;

(7) $(WE_r)_{r \in R} \subseteq W$ is a family of non-empty sets such that WE_r stands for the set of exit states for role $r \in R$;

(8) $\Theta \subseteq W \times W$ is a set of directed edges;

(9) $\lambda : \Theta \longrightarrow L$ is a labelling function, where L can be a timeout, or an illocution schemata and a list of constraints;

(10) $min, Max : R \longrightarrow I\!N$ $min(r)$ and $Max(r)$ return the minimum and maximum number of agents that must and can play role $r \in R$.

At execution time agents interact by uttering grounded illocutions matching the specified illocution schemata, and so binding their variables to values, building up the *scene context*. Moreover, arcs labelled with illocution schemata may have constraints attached, based on the scene context, to impose restrictions on the paths that the scene execution can follow. For instance, in a scene auctioning goods following the English auction protocol, we can specify by means of constraints that buyers can only submit bids greater than the last one (where values are being bound in the scene context).

While a scene defines a particular dialogical environment, the causal, temporal and other content relationships among scenes are expressed through a special type of scene we call *transitions*. The interlacing of regular scenes and transitions is captured through the *Performative Structure* of the EI.

A performative structure can be seen as a network of scenes, whose connections are mediated by transitions, and determines the role-flow policy among the different scenes by showing *how* agents, depending on their roles, may get into different scenes (other conversations), and showing *when* new scenes

(conversations) will be started.

In all EIs we assume that there is always an initial and a final scene, which are the entry and exit points of the institution. We also allow, at run time, to have multiple instances of the same scene.

Technically, we define a *Performative Structure* as follows:

$PS = \langle S, T, s_0, s_\Omega, E, f_L, f_T, f_E^O, C, ML, \mu \rangle$, where

(1) S is a set of scenes;

(2) T is a set of transitions;

(3) $s_0 \in S$ is the *initial* scene;

(4) $s_\Omega \in S$ is the *final* scene;

(5) $E = E^I \bigcup E^O$ is a set of arc identifiers where $E^I \subseteq S \times T$ is a set of edges from scenes to transitions and $E^O \subseteq T \times S$ is a set of edges from transitions to scenes;

(6) $f_L : E \longrightarrow DNF_{2^{V_A \times R}}$ maps each arc to a disjunctive normal form of pairs of agent variable and role identifier representing the arc label;

(7) $f_T : T \longrightarrow \mathcal{T}$ maps each transition to its type;

(8) $f_E^O : E^O \longrightarrow \mathcal{E}$ maps each arc to its type (one, some, all or new);

(9) $C : E \longrightarrow ML$ maps each arc to a meta-language expression of type boolean, i.e. a formula representing the arc's constraints that agents must satisfy to traverse the arc;

(10) ML is a meta-language.

(11) $\mu : S \longrightarrow \{0, 1\}$ states whether a scene can be multiply instantiated at run time or not.

Transitions can be thought of as gateways between scenes or as a change of

conversation. At run-time, transitions can be seen as a type of router in the context of a performative structure. We represent them as a canonical type of scene with arcs incoming from scenes and arcs outgoing to scenes (see the little triangles and half-circles connecting scenes –pictured as boxes– in figure 7). Labels on the directed arcs determine which agents, depending on their roles, can progress from scenes to transitions, or from transitions to scenes.

There are different types of transitions and different types of outgoing arcs. The transition type allows one to express choice points (*Or* transitions) for agents to choose which target scenes to enter, or synchronisation/parallelisation points (*And* transitions) that force agents to synchronise before progressing to different scenes in parallel. On arcs from a transition to a target scene, type determines whether agents following that arc will incorporate into *one*, *some* or *all* of the current executions of the target scene, or if they will go to a *newly* created execution of the target scene.

2.1.2 Norms and Commitments

So far we have dealt with the way interactions within an EI can be defined and organized, we now need to say how they are going to have the intended effect.

We start by noting that an institution (electronic or otherwise) is concerned only by whatever is regulated to happen inside it. Agreements, misconducts, or whatever else that happens outside is in principle disregarded or impertinent to the institution. The main purpose of the institution is to make sure that what happens inside has the effects participants have agreed to.

Actions within an institution, we said, are speech acts. Those speech acts that are made as prescribed by the performative structure of an institution (during an enactment) create obligations or socially binding commitments whose fulfillment is warranted by the institution. We make such intended effects of commitments explicit through what we have been calling *normative rules*.

We define two predicates that will allow us to express the connection between illocutions and norms:

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

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

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 schemata l_j of scenes s_j and s'_j respectively, and e_k, e'_k are boolean expressions over variables from the illocution schemata $\mathbf{i}_{l_j}$ and $\mathbf{i}'_{l_j}$, respectively.

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 schemata and satisfying $e'_1, \ldots, e'_{m'}$, in the corresponding scenes in order to fulfil the normative rule.

Paraphrasing what we have done, the notions we introduce picture the regulatory structure of an EI as a "workflow" (performative structure) of multiagent protocols (scenes) along with a collection of (normative) rules that can be triggered off by agents' actions (speech acts).

Notice that the formalisation of an EI focuses on macro-level (societal) aspects, instead of on micro-level (internal) aspects of agents. Notice also that the whole framework induces an interaction-centered perspective for the design and testing methodologies. Finally notice that the tools we present below and the methodology embedded in them draw substantially from the theoretical framework we just described.

3 An Integrated Development Environment (IDE) for Electronic Institutions

The Electronic Institutions Integrated Development Environment (EIDE) is a set of tools to support the engineering of MAS as electronic institutions. EIDE allows for engineering both electronic institutions and their participating software agents. Notably, EIDE moves away from machine-oriented views of programming toward organisational-inspired concepts that more closely reflect the way in which we may understand distributed applications such as MAS. It supports a top-down engineering approach: firstly the organisation, secondly

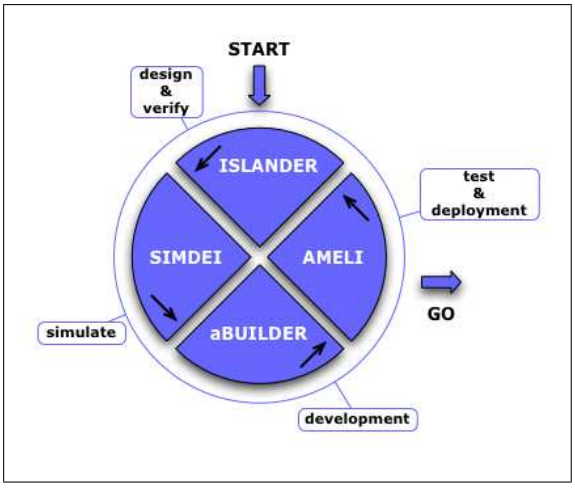

Fig. 1. Electronic Institution Development Cycle

the individuals.

EIDE is composed of:

ISLANDER A graphical tool that supports the specification and verification of institutional conventions. That is, the specification and verification of the EI components presented in section 2.

SIMDEI A simulation tool to animate and analyse *ISLANDER* specifications prior to the deployment stage.

aBUILDER An agent development tool which given an *ISLANDER* specification of an institution supports the generation of agent skeletons for that institution. The generated skeletons can be used on EI simulations supported by *SIMDEI*, or in the real execution of the institution supported by *AMELI*.

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

Monitoring tool A tool which permits the monitoring of EI executions run by *AMELI*. It graphically depicts all the events occurring during an EI ex-

ecution.

Figure 1 depicts the role of the EIDE tools in an electronic institution's development cycle. Notice that such cycle is regarded as an iterative, refining process fully supported by the EIDE tools. In what follows we detail the different steps of such development cycle along with the roles played by the EIDE tools.

3.1 Design

We assume that the development of EIs must be preceded by a precise specification that fully characterise the institutional conventions. In other words, a full specification of EI components as presented in section 2. The analysis required for the complete specification of the system forces the designer to gain a thorough understanding of the modelled institution before developing it. Also, it permits to detect the critical points of the system at an early stage.

The specification of electronic institutions is supported by *ISLANDER* (Esteva et al., 2002). The tool facilitates the work of the institution designer combining graphical and textual specifications of EI components, based on the formalisation of EIs presented in (Esteva, 2003) and outlined in section 2. Graphical specifications facilitate the work of the MAS designer, as well as the understanding of the specification. Concretely, the parts specified graphically are: the conversation protocol in scenes, the relationships among roles in the dialogical framework, and the performative structure graph.

The rest of the elements within an institution are specified textually. The tool tries to structure the way in which the textual information is entered

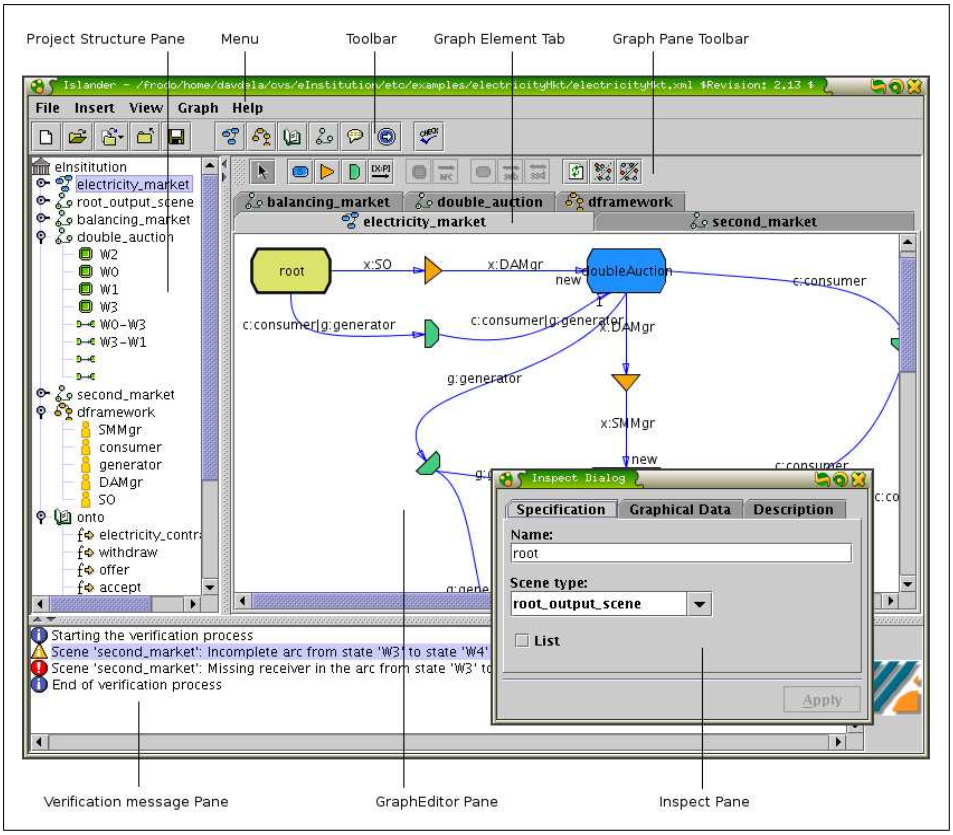

Fig. 2. Islander GUI

to make the work of the designer easier. Thus, for each element the required information is divided into different fields that have to be filled and each field is labelled by a name identifying which information it contains. For instance, when the user has to specify an illocution scheme it is required to fill in a set of fields corresponding to the illocutionary particle, the sender, the receiver, the content expression and the time variable.

Figure 2 depicts the graphical user interface of *ISLANDER*. On the *project structure pane* the user can see all the elements and sub-elements that belong to the current specification, ordered by category. It permits the user to navigate among them. When she 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. That is to say, the edition of the graphical component of performative structures, dialogical frameworks and scenes. 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*.

The result of the design stage is a sound and unambiguous specification of the institutional conventions.

3.2 Verification

Once specified an institution, it should be verified before opening it to external, participating agents. This step is twofold. While the first verification stage focuses on *static*, structural properties of a specification, the second stage is concerned with the expected *dynamic* behaviour of the EI.

The *static verification* of EIs amounts to checking the structural correctness of specifications. This process is fully supported by *ISLANDER*, that performs the following verifications:

- **Integrity.** The tool checks that all 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.
- **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 from each scene agents can always reach the final scene.
- **Protocol Correctness.** It checks that scene protocols are properly 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 properly specified according to the scene dialogical framework.

- **Normative rules' correctness.** It checks that normative rules are specified correctly and that agents can eventually fulfil them. That is, agents can reach the scenes where they have to utter the illocutions for fulfilling each obligation.

The user can activate the verification of the current specification at any time during the design state, and the errors found are shown on the *verification message pane* (see figure 2). Moreover, *ISLANDER* allows the user to move to the context of the error by simply clicking on the error message. When a user selects an error all the panes of the application are modified in order to focus on the elements causing the error. Once an EI has been specified and verified, the user can export the specified institution into XML to be subsequently employed by *AMELI* (see Figure 5).

The *dynamic verification* of EIs, on the other hand, is carried out via simulation. The purpose of the simulation is to study the dynamic behaviour of the specified institution under different circumstances. Simulations of EIs can be run using the *SIMDEI* simulation tool developed over REPAST (Collier, 2003). *SIMDEI* supports simulations of EIs with varying populations of agents to conduct *what-if analysis*. For instance, it would permit the study of the behaviour of an electronic market with different populations of buyers and sellers.

The process starts out with the definition of populations of agents of varying features capable of acting in the specified institution. This process is partially

supported by the *aBUILDER*, which is capable of generating, from a specification, an agent skeleton depending on the agent roles and interactions in which the agent may participate. At this point, we want to remark that an EI specification defines what agents can do depending on their roles but does not contain information on how they have to take their decisions. Hence, agents cannot be completely generated from the specification. In order to completely define an agent, agent designers must fill up the generated skeleton with decision making procedures. Once agents have been implemented, simulations of the EI can be run using the *SIMDEI* simulation tool. The institution designer is in charge of analyzing the results of the simulations and return to the design stage if they differ from the expected ones.

3.3 Development

Once the institution specification has been validated, it is time for agent programmers to implement their participating agents. Recall that among the roles that can participate within an EI, we distinguish between the internal and external roles. Since an EI delegates their services and duties to the agents playing the internal roles (staff agents), their complete development is needed before the EI can be deployed and opened to external agents playing external roles. Notice that we do not impose restrictions on the type of agents that can participate in an EI. Thus, agent designers can choose the language and architecture that are better to fulfil their goals.

Nonetheless, we believe that it is important to support this intricate development process via the *aBUILDER* tool (depicted in figure 3), that supports the graphical specification of agent behaviours starting from an institution's

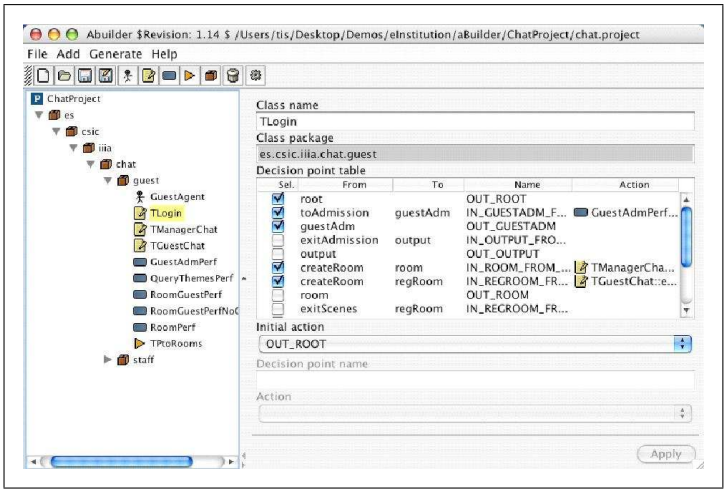

Fig. 3. aBUILDER GUI

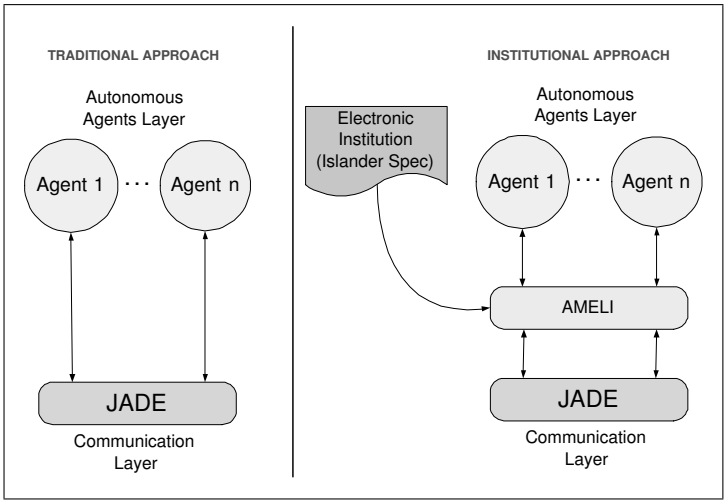

Fig. 4. Agent mediation via electronic institutions

specification created with *ISLANDER*. *aBUILDER* supports the automatic generation of agent (code) skeletons based on graphical specifications of agent behaviours.

The current version of the *aBUILDER* facilitates the development of agents in JAVA in a pre-defined architecture. In the future it is planned to support other languages and architectures.

3.4 Deployment

An electronic institution defines a normative environment that shapes agent interactions. At run time, agents participating in an EI devote their time to jointly start new scene executions, to enter active scenes where they may interact by uttering speech acts, to leave active scenes to possibly enter other scenes, and finally to abandon the institution. As pointed out before, one of the main issues of institutions is the enforcement of its rules on the participating agents. Since participants within an EI may be heterogeneous and self-interested agents, we cannot expect that these agents will behave according to the institutional rules. Therefore, unlike traditional approaches that allow agents to openly interact with their peers via a communication layer, our computational realisation of an EI must be regarded as a *social* middleware that sits between the participating agents and the chosen communication layer validating (filtering in) or rejecting (filtering out) their actions as shown in figure 4[4].

3.4.1 Institution Infrastructure

Our architecture is composed of the following layers:

- **Autonomous agent layer.** Autonomous 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 trans-

[4] In figure 4, Jade(Bellifemine et al., 2001) is employed as the communication layer.

port service.

Notice that participating agents in an institution do not interact directly; they have their interactions *mediated* by *AMELI* through a special type of internal agent called *governor* that is attached to external agents. Through governors, *AMELI* also provides external agents with the information they need to successfully participate in the institution. For instance, information about the participating agents within a scene execution. And more importantly, *AMELI* takes care of the enforcement of the institutional conventions: guaranteeing the correct evolution of each scene execution (preventing errors made by the participating agents by filtering erroneous or unacceptable illocutions, thus protecting the institution); guaranteeing that agents' movements between scene executions comply with the specification; and controlling obligations participating agents acquire and fulfil.

As depicted in figure 5 the current implementation of *AMELI* is a multiagent system itself composed of four 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 agent transitions between scenes. There is one transition manager per transition.
- **Scene manager (SM)**. Responsible for governing a scene execution. There is one scene manager per scene execution.
- **Governor (G)**. Each one is devoted to mediating the participation of an external agent within the institution. There is one governor per participating

agent.

And what are the features of *AMELI*?

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. In order for agents to communicate with their governors, they are solely required to be capable of opening a communication channel. Since no further architectural constraints are imposed on external agents, we can regard *AMELI* as *agent-architecture neutral*.

Observe that *AMELI* is a general-purpose platform in the sense that the very same infrastructure can be used to deploy different institutions. With 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. Therefore, it must be regarded as *domain independent*, and it can be used in the deployment of any specified institution without any extra coding. During an EI execution, the agents composing *AMELI* keep the execution state and they use it, along with the institutional rules encoded in the specification, to validate agents actions and to asses their consequences.

As depicted in figure 4, the infrastructure is divided into two layers: *AMELI* and a communication layer offering a reliable and orderly transport service. In this manner, *AMELI* agents do not need to deal with low-level communication issues, and therefore focus on handling the institution execution. The current implementation of the infrastructure can either use JADE (Bellifemine et al., 2001) or a publish-subscribe event model as communication layer. When em-

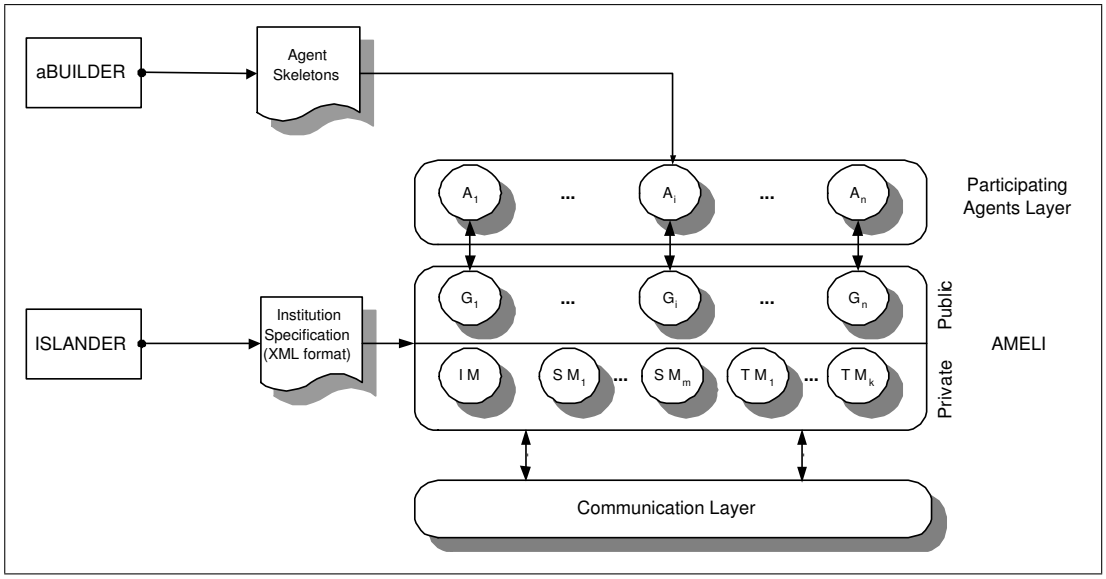

Fig. 5. Electronic institution architecture

ploying JADE, the execution of *AMELI* can be readily distributed among different machines, permitting the *scalability* of the infrastructure. Finally, participating agents regard our architecture as *communication neutral* since they are not affected by changes in the communication layer.

3.4.2 Electronic Institution Execution

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.

An EI execution can be monitored thanks to the *monitoring tool* that depicts all the events occurring at run time. Fairness, trust and accountability are the

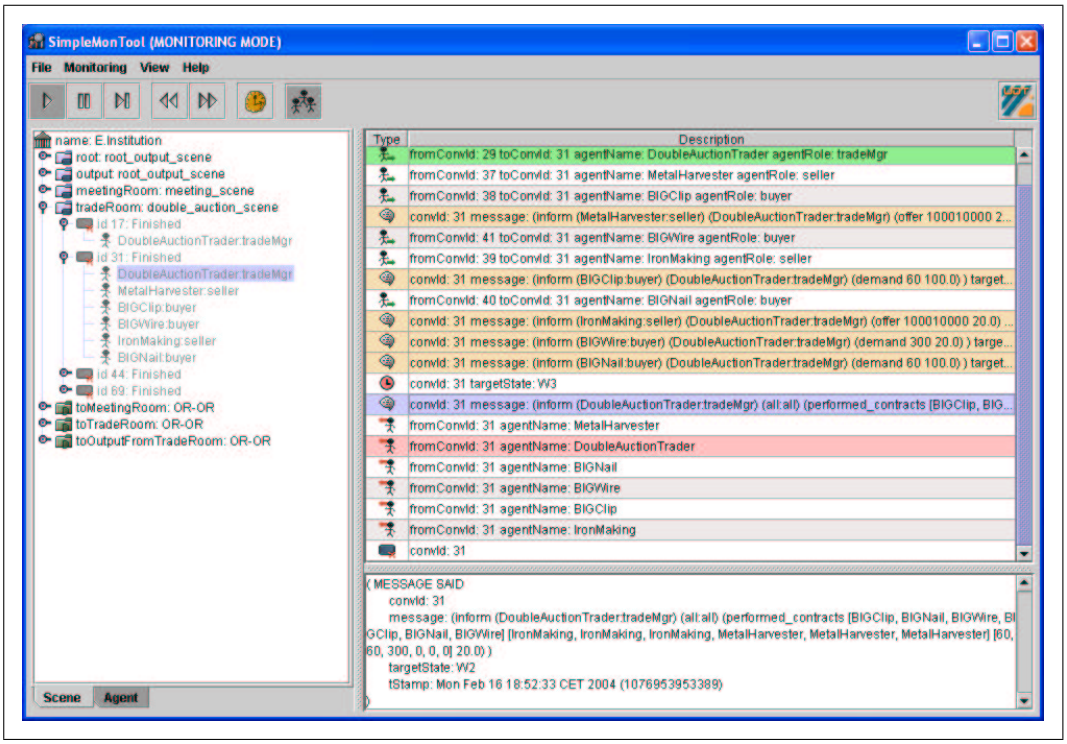

Fig. 6. Monitoring of an electronic institution execution

main motivations for the development of a monitoring tool that registers all interactions admitted in a given enactment of an electronic institution (Noriega, 1997; Rodríguez-Aguilar, 2001). 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 and improve it. From the point of view of the institution designers, the tool could serve to test 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 document fraudulent behaviours of external agents.

The monitoring tool (see figure 6) displays all the interactions occurring in the different scene and transition executions, along with agents' movements among scenes. The tool shows on its left side the list of scene and transition

executions. When the user selects one of them, the tool shows on its right side all the events that took place during the execution. That is, the messages exchanged, the entrance of new agents and the departure of any of the participants. Moreover, the tool also allows monitoring the participation of an agent in an EI execution. In this case, it shows the scenes in which the agent has taken part, along with the messages that it has exchanged. Finally, the tool also displays the inadmissible actions that agents attempt. Those that do not comply with the institutional conventions encoded in the specification and the current execution state. For instance, when an agent tries to utter an illocution which is not correct with respect to a scene execution.

4 Case study

In the Mediterranean coast, fresh fish has been traditionally sold through downward bidding auctions operating in auction houses in fishing towns. Fish is presented in collections of boxes, called *lots*, and put up for auction following a Dutch-like protocol: price is progressively and quickly lowered —4 quotes per second— until a buyer submits a bid or the price descent reaches the reservation price. The buyer submitting the bid can decide to buy the complete lot or just some boxes. In the later case, the remaining boxes are put back up for auction in the next round. When the last box of the last lot is sold, the auction is over.

Some fishmarkets are adapting their selling methods to new technologies and most auctions are nowadays somewhat automated, although the presence of human buyers in the auction houses is still necessary. This has two significatn drawbacks. First, it restricts potential buyers to those present in the auction

house. Second, it makes simultaneous participation in several auctions costly, since companies have to send a representative to each one. The elimination of such limitations would be profitable for both buyers and sellers. Increasing the number of buyers makes the market more competitive, and thus increases the buying price to the benefit of sellers. It also permits the participation of buyers without intermediaries saving costs to the buyers.

Agent technologies may be used to eliminate these limitations. The Multiagent System for FIsh Trading (MASFIT)(Cuní et al., 2004) [5] allows buyers to remotely and simultaneously participate in several wholesale fish auctions with the help of software agents, while maintaining the traditional auctions as they are. The participation of buyer agents in auctions is mediated by an EI. MASFIT interconnects multiple auction houses, hence structuring a federation of them. Significantly, MASFIT guarantees that buyer agents have access to the same information, and have the same bidding opportunities as human buyers physically present at the auction house. Furthermore, the system does not alter the current operation of the auction houses.

In order to permit buyer agents to participate in the auctions, the auction systems running in the (physical) auction houses have been extended to connect with the electronic institution. Thus, the auction systems send to the MASFIT electronic institution information about all the events occurring at the corresponding auction house that are relevant for the buyers. These are: the registration of new goods and the information about each auction round. MASFIT receives this information and passes it to the buyer agents. Moreover, it controls buyer agents access to the auctions, and collects their bids during the auctions. MASFIT sends to the auction systems information about

[5] http://www.masfit.net.

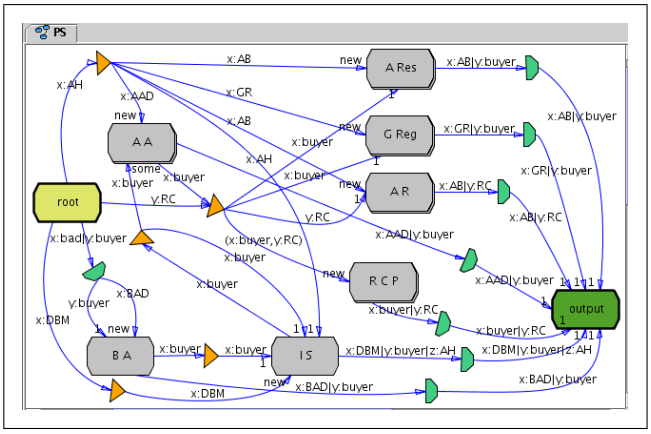

Fig. 7. Specification of the MASFIT institution performative structure.

which buyer agents are taking part in their auctions and the bids that they submit.

Since each auction house wants to continue having control over which buyers are authorized to participate in their auctions, human buyers must register in each auction house before this auction house authorizes buyer agents to participate on their behalf. As a consequence, buyer agents may not have access to all the federated auction houses. For this reason, we distinguish between the agents' admission to the institution, as the federation of auction houses, and their admission to the different auction houses. Hence, the MASFIT's institution keeps track of which agents are taking part in each auction house. Notice, that only the agents admitted in an auction house must receive the information about their auctions.

As reported in (Cuní et al., 2004), the EIDE tools played a key role in the design and development of the MASFIT system. On the one hand, the MASFIT electronic institution was specified using *ISLANDER*. On the other hand, agents in the institution have their interactions mediated by *AMELI*.

Figure 7 depicts the specification of MASFIT's performative structure cre-

ated with the aid of *ISLANDER*. It is composed of a large number of scenes representing different activities in MASFIT, namely: *buyer admission* (BA), where a *buyer admitter* controls buyer agents' access to the federation; *info-seeking* (IS), where buyer agents are informed about which auction houses are federated and can access historical data about them; *auction admission* (AA), where an *auction admitter* controls buyer agents' access to an auction house; *good registration* (GReg), where the *good register* agent informs the buyers of the lots registered within an auction house; the *auction room* (AR), where an *auction broker* mediates the participation of RCs (on behalf of buyer agents) in an auction house; *auction results* (ARes), where buyer agents are informed about the results of an auction; and the *RCprogramming* (RCP), where a buyer agent programs —through dialogue— a RC [6] to participate in an auction.

Since MASFIT is a federation of auction houses, there are some scenes devoted to common activities (federation level), while others are devoted to the activities of a particular auction house (auction level). The scenes at the federation level are: *root*, *output*, *buyer admission* (BA) and *info-seeking* (IS) scenes. There is one single execution of these scenes when the institution is running. The rest of the scenes mediate buyer agents' participation in particular auctions. There is one execution of each of these scenes per auction house connected to the federation, except for the *RCProgramming* scene. In this case, there is one execution of the *RCProgramming* scene per buyer agent admitted within an auction house.

Buyer agents must go first to the *buyer admission* scene where the buyer admitter controls their access to the federation. If they are admitted, they

[6] An RC is a simple agent that actually performs the bidding.

can move to the *info-seeking* scene where they can request historical data. Moreover, they are also informed about the auction houses connected to the federation. From the *info-seeking* scene, buyer agents can try to enter the different auction houses by moving to the corresponding *auction admission* scenes.

Finally, notice that the participation in a market scenario as created by MASFIT, a federation of auction houses, is a complex decision-making task, as buyer agents are participating simultaneously in several auctions (A.Byde et al., 2002). Buyer agents receive information from different auction houses and they should decide the most suitable place to buy. Agents have to manage huge amounts of information -even uncertain information- and their reasoning and processing time must be short enough to react to changes. To support this complex design, MASFIT makes available tools to create, customise, manage and train software buying agents. Human buyers are provided with an agent skeleton that they may customise to their buying preferences using available tools. The customization is done by means of buying lists, strategies and logistics. The buying lists contain the products that the buyer wants to buy. In the case of strategies, they are predefined but depend on some parameters that must be set by the user. Finally, logistics include information about transportation costs from the different auction houses to the user storehouse. Thereafter, the buyer agent uses all this information to bid at the auctions. When it receives information about the registration of a new lot, it checks if the product is required in its buying list. If so, the agent determines a price to bid and the quantity it wants to acquire.

5 Related Work

Recently, a number of MAS methodologies - e.g. GAIA (Zambonelli et al., 2003), Tropos (Giunchiglia et al., 2001), or (Sturm et al., 2003) 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 coabs (2004) and FIPA compliant platforms such as JADE (Bellifemine et al., 2001) 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 (Hanachi and Sibertin-Blanc, 2004) 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 (Zambonelli et al., 2003)).

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. (Parunak and Odell, 2002),(Dignum, 2004), (Vazquez and Dignum, 2003)), 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. (Omicini et al., 2004)). 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 [7]. 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.

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

6 Closing Remarks

The design and development of multiagent systems is a complex and difficult task. This complexity arises from their distributed nature and from the need of having high level and flexible interactions among autonomous entities (Jennings et al., 1998). This is specially true in the case of open MAS populated by heterogeneous and self-interested agents. We propose that this complexity can be handled by introducing regulatory structures as EIs. Specifically, we defend that open MAS can be effectively designed and developed as electronic institutions (Noriega, 1997; Rodríguez-Aguilar, 2001; Esteva, 2003). Similarly to human institutions, EIs define the rules of the game in agent societies, by fixing what agents are permitted and forbidden to do.

Due to the complexity of EIs, we early identified the need of software tools that support EIs design and development as one of the main issues in our research (Noriega, 1997; Rodríguez-Aguilar, 2001; Esteva, 2003), and in this paper we have introduced an integrated development environment for the engineering of multiagent systems as electronic institutions as the result of our group's effort over these years. Notice that in the context of EIs, we differentiate between the rules and the players (agents). However, we believe that is is important to support the development of both. Therefore, EIDE groups a set of tools which provide support to the specification, verification and execution of institutional rules, as well as the development of participating agents. Specifically, *ISLANDER* supports the specification of EIs as formalised in (Esteva, 2003) and their static verification, *SIMDEI* supports their dynamic verification, while *AMELI* gives support to their execution by facilitating agent participation on the institution, while enforcing the institutional rules encoded

in the specification. Moreover, the monitoring tool permits to monitor EIs executions performed by means of *AMELI*. Complementary, the *aBUILDER* tool partially supports the development of agents for a given EI.

Major benefits derive from employing the EIDE tools. Firstly, they help shorten the development cycle. The engineering of an electronic institution entails a low-cost implementation since only its participating agents must be programmed. The inherent flexibility of *ISLANDER* in the design of coordination mechanisms favours an easy, ready maintenance: when changes are accommodated in a new specification, they are ready to be run by *AMELI*, and agents are ready to plug and play. Secondly, the EIDE simulation tools support what-if analysis of electronic institutions' designs prior to their deployment, facilitating the location of unexpected behaviours that may jeopardise critical applications. Furthermore, agent developers are also supported by *aBUILDER*.

EIDE has proven to be highly valuable in the development of e-commerce applications such as the MASFIT system presented in section 4. However, a wider range of application areas may be tackled with the aid of the EIDE tools. In general terms, the electronic institutions approach is deemed as appropriate in complex domains where multiple partners are involved, and a high degree of coordination and collaboration is required. Thus, electronic institutions to support workflow management, the monitoring and management of shop-floor automation, or supply network management issues look promising in the near future.

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

7 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

A.Byde, Preist, C., Jennings, N., 2002. Decision procedures for multiple auctions. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems. pp. 613–620.

Bellifemine, F., Poggi, A., Rimassa, G., 2001. Developing multi-agent systems with jade. In: Castelfranchi, C., Lesperance, Y. (Eds.), Intelligent Agents VII. No. 1571 in Lecture Notes in Artificial Intelligence. Springer-Verlag, pp. 89–103.

coabs, 2004. Control of agent based systems. http://coabs.globalinfotek.com.

Collier, N., June 2003. Repast: An extensible framework for agent simulation. http://repast.sourceforge.net.

Cuní, G., Esteva, M., Garcia, P., Puertas, E., Sierra, C., Solchaga, T., August 2004. Masfit: Multi-agent systems for fish trading. In: 16th European Conference on Artificial Intelligence (ECAI 2004). Valencia, Spain, pp. 710–714.

Dignum, V., 2004. A model for organizational interaction. Ph.D. thesis, Dutch Research School for Information and Knowledge Systems, iSBN 90-393-3568-0.

Esteva, M., 2003. Electronic Institutions: from specification to development. IIIA PhD Monography. Vol. 19.

Esteva, M., de la Cruz, D., Sierra, C., July 15-19 2002. Islander: an electronic institutions editor. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2002). Bologna, Italy, pp. 1045–1052.

Esteva, M., Rodríguez-Aguilar, J. A., Sierra, C., Arcos, J. L., Garcia, P., 2001. On the formal specification of electronic institutions. In: Sierra, C., Dignum, F. (Eds.), Agent-mediated Electronic Commerce: The European AgentLink Perspective. No. 1991 in Lecture Notes in Artificial Intelligence. Springer-Verlag, pp. 126–147.

Giunchiglia, F., Mylopoulos, J., Perini, A., November 2001. The tropos software development methodology: Processes. Tech. Rep. 0111-20, ITC-IRST.

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

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

Jennings, N. R., Sycara, K., Wooldridge, M., 1998. A roadmap of agent research and development. Autonomous Agents and Multi-agent Systems 1, 275–306.

Noriega, P., 1997. Agent-Mediated Auctions: The Fishmarket Metaphor. IIIA Phd Monography. Vol. 8.

North, D., 1990. Institutions, Institutional Change and Economics Perfomance. Cambridge U. P.

Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., Tummolini, L., July 19-23 2004. Coordination artifacts: Environment-based coordination for intelligent

agents. In: Third International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS'04). New York, USA, pp. 286–293.

Parunak, H., Odell, J., 2002. Representing social structures in uml. In: Agent-Oriented Software Engineering II. LNCS 2222, springer-verlag Edition. pp. 1–16.

Rodríguez-Aguilar, J. A., 2001. On the Design and Construction of Agent-mediated Electronic Institutions. IIIA Phd Monography. Vol. 14.

Rodríguez-Aguilar, J. A., Noriega, P., Sierra, C., Padget, J., 1997. 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). pp. 207–224.

Sturm, A., Dori, D., Shehory, O., 2003. Single-model method for specifying multi-agent systems. In: Proceedings of AAMAS 03. Melbourne, Australia, pp. 121–128.

Vazquez, J., Dignum, F., 2003. Modelling electronic organizations. In: Multi-Agent Systems and Applications III, springer-verlag Edition. Vol. 2691 of LNAI. pp. 584–593.

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