CHARMS: A Charter Management System. Automating the Integration of Electronic Institutions and Humans

Ismel Brito, Nardine Osman, Jordi Sabater-Mir, and Carles Sierra

Artificial Intelligence Research Institute (IIIA)
Spanish National Research Council (CSIC)
Bellatera, Barcelona, Spain

Abstract. The execution of process models is usually presented through a graphical user interface, especially when users' input is required. Existing mechanisms, such as Electronic Institutions (EIs), provide means to easily specify and automatically execute process models. However, every time the specification is modified, the graphical user interface (GUI) needed during the execution stage should be manually modified accordingly. This paper proposes a system that helps maintain such GUIs in an efficient and automated manner. We present and test Charms, a system built on top of EIs that allows the automatic creation and update of GUIs based on the provided process model specification.

1 Introduction

The issue addressed by this paper is that of *specifying*, *executing*, and *maintaining* process models. The term 'process model' is a very generic term that has been used in various contexts, from business processes, to concurrent communicating systems. In all of these cases, the goal of a process model is usually to understand and define how a given process works. Several languages have emerged for the specification of processes. For instance, UML diagrams may be used to specify processes in an easy graphical manner. Others approaches have focused on the analysis of processes; for instance, process calculi permit the formal reasoning over process models [Cardelli and Gordon, 1998,Milner, 1999]. This was followed by executable languages that would allow the execution of processes in an automated manner (e.g. BPEL [Boile et al., 2006]). This paper tries to extend this line of work by providing means for *maintaining* processes in a straightforward manner that appeals to the average user.

We note that several existing tools already provide users with means for specifying and automatically executing their processes (e.g. BPMN). This paper, however, states that in addition to the specification and execution of processes, a management system is required for maintaining these processes models. We argue that some models need to evolve with time, and this evolution should not imply the redesign of the specification and its corresponding user interface from scratch, but it should permit a more automated evolution in which the

user interface automatically adapts to fit the updates and modifications. For example, take the process model describing the process of paper submission and review. With current conference management systems, such as ConfMaster (`http://www.confmaster.net/`) and EasyChair (`http://www.easychair.org/`), the user interface is hard-coded to suit the predefined traditional process. But what if the entire peer review process is changed into something more open and similar to the Interdisciplines conferences (`http://www.interdisciplines.org/`), where authors are invited to submit papers (skipping the review process) and papers are simply discussed online by other researchers in the field over a predefined period of time? Such a change in the conference's process model, or what we refer to as charters, would require a radical redesign of the entire web-based interface to suit the changed model. Sometimes even minor changes in the model could require major modification to the user interface.

In this paper, we do not wish to redesign a system from scratch for the specification, execution, and management of process models from scratch. Instead, we adopt an existing powerful one, electronic institutions (EIs), which is used for the specification and execution of process models. We then propose a management system that may be built on top of EIs. In summary, the concrete goal of this paper is to provide a mechanism that helps maintain process models by eliminating the need for manually modifying the user interface every time the model is modified. This is done by introducing the automated generation and update of interfaces based on the specification of the model. This, in essence, provides a user friendly approach for maintaining process models.

However, before we describe the paper's main contribution, we first need to provide a clearer introduction to process models, which the remainder of this paper refers to as charters. This is achieved by Section 2. The chosen system used to specify and execute charters, the Electronic Institutions, is introduced in Section 3. Moreover, before moving to our main contribution, Section 4 briefly analyses the differences between EIs and traditional modelling languages explaining why we think that EIs are a better option to specify and implement charters.

The main proposal, the Charms system which permits the management of charters by automating the user interface (UI), is presented in Section 5. This is followed by a description of a real conference model example in Section 6, which provides an overview of our motivating example: the Interdisciplines conference. Section 7 then explains how the Charms system is applied to this example. Finally, we conclude with Section 9.

2 Charters: Defining Process Models

According to the Oxford English dictionary, a charter is "a written evidence, instrument, or contract executed between man and man", it is a document "granting privileges to, or recognizing rights of, the people, or of certain classes or individuals".

We say a charter is essentially a convention, agreed upon by a set of participants, that sets the rules of the interaction between these participants by declaring the rights and privileges each participant has. Accordingly, participants are legally binded to obey and observe these rules, their contract.

In its most basic form, a charter $\mathcal{C}$ may be composed of a set of rights $\mathcal{R}$, where a right is defined as the tuple $\langle o, e, \mathcal{D} \rangle$, and o represents the object that the entity e has a right to, and $\mathcal{D}$ represents the set of conditions under which this right holds. For example, $\langle vote, citizen, \{is\_adult(citizen)\} \rangle$ states that a citizen has a right to vote if it is an adult.

However, we note that the rights of one participant usually imply the obligation or forbiddance of another participant to perform some action(s). For example, the right to education implies an obligation on the governments to provide education to their citizens. As such, deontic systems have been used to specify the "rights and duties" of individuals and organisations. In a deontic logic, a system is specified by a set of permissions, obligations, and related concepts. However, it is interesting to note that one may pick only one of the deontic concepts to be the basic one, and then all the remaining concepts may be defined in terms of the basic one by making use of negation. For example, if we assume 'obligations' to represent the basic deontic concept, then the 'permission' to smoke is equivalent to the lack of obligation to not smoke.

In organisations and institutions, charters are used to specify the set of declarative rules (or norms). An example of such a declarative rule is that reviewers should provide their reviews on time. However, in practice, the need arises for concrete details on who may perform what action, when to carry out that action, and under what conditions (where the conditions are usually context dependent: different states of the interaction require different conditions). As we move from simple human interactions to more complex ones, enforcing the rules and guaranteeing an entity's right becomes a challenge. For this, bylaws are drafted under the authority of the charter. We think of bylaws as the set of procedural rules, as opposed to declarative ones. For example, instead of simply stating that reviewers should provide their reviews on time, the procedural rule could state that after a reviewer receives a paper, they should submit their review in one month, at the latest. This becomes especially useful in distributed open systems that are composed of autonomous interacting agents, where one common approach is to make use of a more precise plan of action — as opposed to relying on a set of declarative rules — to guide the agent's actions, yet enforcing the norms in an efficient manner and guaranteeing the rights of participants. This plan of action is usually defined through a flow graph. A flow graph essentially specifies the order in which the steps must be executed which steps could be executed in parallel, etc. The entire flow graph should be consistent with the norms and their requirements, and it could aid penalising misbehaving participants.

Formally, a bylaw may then be defined as tuple: $\mathcal{B} = \langle S, R, A, T \rangle$, where S represents the set of states, R is the set of roles that users may play in the interaction, A is the set of actions that may be performed, and $T : S \times A \times R \rightarrow S$ is a transition function that defines which role can perform which action at which

state, and the new state resulting from that action. The tuple $\mathcal{B}$ is analogous to Kripke structures and finite state machines.

We note that the set of rights $\mathcal{R}$ specified by the charter $\mathcal{C}$ should map with the transitions of T (the transitions follow the rights $\mathcal{R}$). As such, it is necessary to be able to verify that a specific bylaw satisfies its charter ($\mathcal{B} \models \mathcal{C}$). This issue is outside the scope of this paper, but we note that formal verification mechanisms (such as theorem proving, SAT solvers, or model checking) may then be used to help verify that the bylaws do satisfy their charters.

Of course, inheritance of rules should also be studied in the cases of related organisations. For example, an organisation can have a charter/bylaw, and one department in this organisation may have its own charter/bylaws. In such cases, their needs to be a clear study on which rules are inherited, which are overriden, etc. However, this is outside the scope of this paper.

We also note that, in charters and bylaws, potential participants (humans, organisations, or autonomous agents) are grouped into roles according to their responsibilities. A role may be played by several participants (such as the case of having several bidders in an auction system), and a participant may play more than one role (for example, in a stock exchange scenario, a participant may play the role of the buyer of one stock and the seller of another). As such, charters and bylaws do not focus on participants *per se*, but on the roles played by participants. This makes Electronic Institutions, which are introduced by the following section, very appealing to use for specifying bylaws as they already incorporate these concepts.

In the remainder of this paper, we focus on the automation of the GUI of a specific bylaw in execution. We note that references to charters in the remainder of this document imply the procedural rules of charters, or the bylaws.

3 Electronic Institutions

The concept of *electronic institution* (EI) is inspired from human institutions. In open multi-agent systems, where autonomous entities interact to achieve individual goals, one cannot guarantee what will be the final outcome of this interaction. Therefore, and similarly to what happens in human societies, you need mechanisms to guarantee the good functioning of the system despite the local behaviours, which is achieved by enforcing norms and penalising those that misbehave. The use of an electronic institution that regulates the behaviour of agents the same way human institutions regulate the behaviour of people is one of these mechanisms.

There are few concepts that have to be defined to fully understand the electronic institution machinery [Sierra et al., 2004].

- *Agents and Roles*: agents are the players in an electronic institution, interacting by means of the exchange of illocutions, whereas roles are defined as standard patterns of behaviour. Any agent within an electronic institution is required to adopt at least one role although usually it will adopt several.

Some agents play what are called *institutional roles* that are roles that help the good functioning of the institution (like for example the auctioneer in a virtual auction).

- *Dialogical Framework*: EIs establish the acceptable speech acts an agent can utter by defining an ontology and a common language for communication and knowledge representation. Both are bundled in what is called a dialogical framework. In an EI, all the actions are performed through speech acts.
- *Scene*: Interactions between agents are articulated through agent group meetings (which are called scenes) with a well-defined protocol. Protocols in a scene are considered to be the specification of the possible dialogues the participating agents may have.
- *Performative Structure*: Scenes can be connected, composing a workflow, in a so-called performative structure. The specification of a performative structure contains a description of how agents can legally move from scene to scene by defining both the pre-conditions to join and leave scenes. Transitions, a special type of scenes, allow synchronization among agents. They can be used also as choosing points where agents can decide which path to follow in the performative structure, and parallelization points that allow agents to be sent to more than one scene.

Figure 5 shows the *performative structure* we will use in our example. It will be extensively described in section 7. We can see the different scenes (*begin*, *Admission*, *CreateConf*, *RunConf*, *Submission*, *end*) that are connected by means of the *transitions*. The transitions impose restrictions in terms of which roles can follow them. For instance, only an agent playing the role of *admin* can go from the *Admission* scene to the *CreateConf* scene. Figure 6 shows the protocol that the agents in the *Submission* scene have to follow. In the figure you can see the different states and the transitions. It is important to notice that all the agents present in the scene are in the same state of the protocol and all together move to a different state when the right speech acts are uttered.

The link between an agent and the EI is the *governor*. The governor serves the purpose of safe-guarding the EI, i.e. it checks whether a particular message is allowed to be said (and therefore an action to be performed) at the current stage depending on the role of the agent, the state of the scene, etc.

As illustrated in figure 1, the software of the electronic institution provides two APIs. One is used by the governor to provide the agent with the messages coming from the electronic institution (for example, changes in the state of a scene where the agent is participating) or from the governor itself (for instance, saying that a given speech act is not allowed given the current state). The other provides the agent with the mechanism to interact with the governor. For instance, the agent can query the governor about the electronic institution (for example regarding the state of a scene) and utter speech acts. The governor filters the messages, and it either performs the utterances in the electronic institution if they conform with the current state or, communicates the error to the agent. Of course, both the governor and the APIs that allow the communication with the electronic institution are provided by the institution itself.

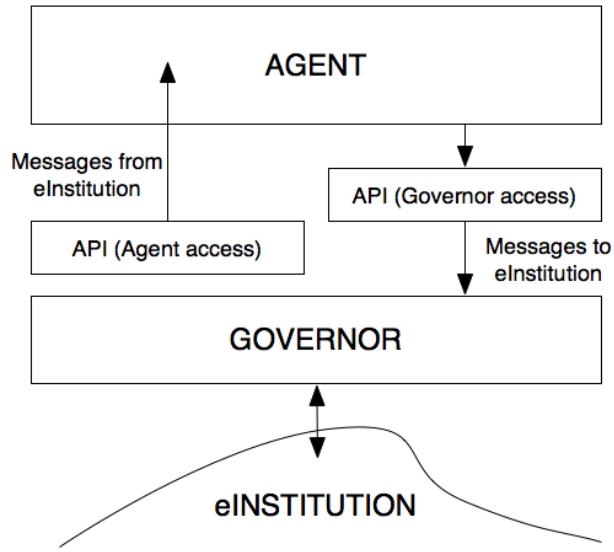


Fig. 1. Governor

The Electronic Institution Development Environment (EIDE)[1] is a set of tools aimed at supporting the engineering of multiagent systems through electronic institutions. EIDE is composed of the following tools:

- ISLANDER: A graphical tool that supports the specification of the rules and protocols in an electronic institution. The diagrams showed in figure 5 and figure 6 have been designed using ISLANDER.
- AMELI: A software platform to run electronic institutions specified using ISLANDER. Using the XML file generated by ISLANDER, AMELI runs a real multi-agent system that implements the specification of the electronic institution. The platform provides the infrastructure for the performative structure and the different scenes with the different protocols as well as the governors. The agents that are going to participate in the electronic institution can then attach to the governors and start the interaction.
- aBUILDER: An agent development tool. Although any agent that can use the APIs provided by the governors can connect with the electronic institution, EIDE provides a graphical tool to specify agents that can be used in that specific electronic institution. aBUILDER uses the XML specification from ISLANDER to build a code template of an agent ready to participate in the electronic institution. The user only has to fill the gaps in the code with the specific actions to be taken in the different scenes and transitions.
- SIMDEI: A simulation tool to animate and analyze electronic institutions. The analysis of a real electronic institution implemented as a multi-agent

[1] http://e-institutions.iiia.csic.es/

system with concurrent threads and agents running in parallel is very difficult. SIMDEI is a tool that simulates an electronic institution (specified also using ISLANDER) following a sequential approach so the debugging and analysis is much more easier. The security aspects are relaxed (for example the figure of the governor disappears) so it is assumed that the the system is not open and that the agents are under control. This simplification in terms of communication makes the execution much faster.

The process of creating and running an specific electronic institution consists of the following steps:

- Design: Electronic institutions can be graphically specified with the aid of ISLANDER [Esteva et al., 2002]. It allows for the definition of a common ontology, all the interactions that agents may have, and the consequences of such interactions.
- Verification: The static verification of EIs amounts to checking the structural correctness of the specifications. This process is fully supported by the ISLANDER editor. On the other hand, the dynamic verification of EIs is carried out via simulation using the SIMDEI simulation tool.
- Development: Once the institution specification is validated it can be deployed and opened for agent participation. Thus it is time for agent programmers to implement their participating agents. Notice that there exist no restrictions on the type of agents that can participate in an EI, except that they have to be able to connect to a governor. The aBUILDER tool can be used for this task.
- Deployment: Since agents may be heterogeneous and self-interested we cannot expect that they behave according to the institutional rules. Therefore any EI is executed via AMELI [Esteva et al., 2004] that mediates and facilitates agents' interactions while enforcing the institutional rules. The execution of an EI starts out by running AMELI after loading the specification. Thereafter, external agents may enter the institution to interact with other agents through AMELI.

Although at first sight EIs and process modelling languages might seem closely similar, there are some fundamental differences that make EIs much suitable for the specification and implementation of charters. In the next section we explain these differences.

4 Comparing EIs to Process Modelling Languages

There are several similarities and differences between business process modelling languages (e.g. BPEL [Boile et al., 2006], or the modelling framework BPMN[2]) and EIs. Traditional workflow languages cover both orchestration (the combination of components/services to constitute the internals of a process[3]) and

[2] http://www.bpmn.org/
[3] In BPMN terminology, this is known as an organisation.

choreography (how different processes interact and exchange information). Electronic institutions concentrate only on choreography as they are neutral with respect to the programming of the individual agents that are considered autonomous and externally programmed, while constraining their public activity by the definition of a very well structured choreography. Before we proceed, we first note that the concept of workflow is general and could be conceived for any type of coordination between processes and agents (e.g. [Jennings et al., 1996,Shepherdson et al., 1999]).

The choreography in BPMN, modelled as messages between processes in the workflow of the agents, does not have a clear structure in the form of a dialogue following a protocol; it consists of individual messages connecting internal processes of the agents. In other words, its structure is less precise and less abstract than in electronic institutions where the choreography happens within the boundaries of a scene[4], as a clearly defined dialogue among a number of agents disconnected from the agent internal structure, and in the network of scenes that allows for sophisticated interconnections between dialogues. A major difference between the two choreographic approaches is that in the case of BPMN the connectors between activities indicate decision points made by a single agent (as the workflow in fact corresponds to a single agent) while in EIs the transitions correspond to synchronisation and parallelization points for agents. The flow of data is perhaps more clearly represented in BPMN than in EIs as in the latter formalism the data updates are performed as a consequence of the choreographic actions, i.e. as a consequence of the exchange of messages among agents within a scene, and written in an action language without any graphical help to the designer. The notion of group in BPMN resembles the notion of scene in EI but the flow of agents between scenes does not have a clear equivalent in the group concept as they cannot be interconnected. The concept of group is more an annotation element than a fundamental operational one as is the case in EIs. Both approaches are neutral with respect to methodology and admit different ways of generating a specification (e.g. LOVeM^{TM} or IDEF for BPMN or PROMETHEUS+ [Sierra et al., 2007] for EI). In summary, the emphasis of EIs is on the flow of agents between activities and the choreography of their activities, while the emphasis of BPMN is on the coordination of the internal activities of the agents and on the sharing of data among these internal processes. Choreography is more structured and abstract in electronic institutions.

The fact that the most important aspect of a charter is to represent *joint* activities of the participants in a process (business or scientific), makes us think that EIs are more suitable for our purposes, given the emphasis on regulated scenes (precisely the notion of joint activity) and the existing tools to design and verify the flow of agents between them. The notion of group in BPMN is too weak and there is no direct equivalent to a flow between groups which is key in the specification of charters.

[4] This is similar to the notion of ambient in ambient calculus [Cardelli and Gordon, 1998]

In summary, it is important to note that the literature is full of very similar and competing approaches on how to model processes: from simple workflows to state diagrams and finite state machines. The list of models is massive: Kripke structures, UML, Petri net, etc. And some approaches, like Statemate [Harel and Politi, 1998], allow both the specification and testing of models. However, what is particularly interesting about EIs (other than providing a simple graphical interface for both the specification and execution of processes) is that their specification is closer to the specification of organisational bylaws ($\mathcal{B} = \langle S, R, A, T \rangle$, as presented by Section 2), which unlike other approaches, provides a clear presentation of the organisational hierarchical structure of the roles of agents and their permitted actions, in addition to a clear specification of the social norms. This makes EIs more suitable for describing institutions and organisations, as opposed to more general processes, such as the traffic light system.

5 Charms: a Charter Management System

The charter management system, Charms, is an independent-domain platform built on top of the EI technology, and aims at specifying and executing charters in a fully automated manner. Although Charms is domain independent, the rest of this paper will use the conference management systems domain as a prototypical domain.

Existing examples of conference management systems are the well known ConfMaster[5] and EasyChair.[6] These systems are hard-coded and therefore, their web interfaces require a manual update every time a change occurs in the specification. They assume that you will manage a conference that follows the standard workflow. But what happens if you want to run a different kind of conference? These systems become inappropriate as the changes to adapt to different workflows are too costly.

Charms wants to overcome this problem. The objectives of the proposed system are twofold:

- to provide charter organisers with straight forward means for the specification and automatic execution of charters, and
- to provide the automatic creation and update of a web-based user interface. The main goal is to eliminate the need for manual modifications in the UI every time the charter changes.

Charms enacts charters as electronic institutions. As we have seen in Section 3, the ISLANDER editor for EIs already provides an easy to use user interface for the formalisation of process models through a simple drag-and-drop mechanism. This formalisation includes the specification of interaction protocols, illocution schemas and a concrete ontology. Once the charter's flow is specified as an EI,

[5] http://www.confmaster.net/
[6] http://www.easychair.org/

Charms executes it using AMELI, also provided by EIDE. However, what is truly novel here is that Charms generates the web-based user interface that allows the users to interact with the specified charter in a fully automated manner.

As illustrated by Figure 2, Charms architecture is based on the standard client-server distributed model. The server side of the Charms architecture relies on two components: an electronic institution (EI) and a Java servlet (or I-Agent). Whilst the EI (the kernel of the system) aids us in specifying and executing charters, the servlet is in charge of notifying the user of relevant events occurring inside the EI. By contrast, the client side of the Charms architecture includes a web-based client application (AJAX) and a web browser (or the UI). Concerning the client application, we have adopted an approach based on two web technologies: Java Server Pages (JSP) and Asynchronous JavaScript and XML (AJAX). JSP technology provides a simplified fast way to create dynamic web content by merging HTML with Java. In contrast, AJAX provides a high level of interactivity, avoiding the undesirable reloading of the web pages after each user action. Hence, the client application only needs a relatively recent web browser and therefore it is not necessary to install any special software. We note that in practice, there is one client side per each user connected to the platform.

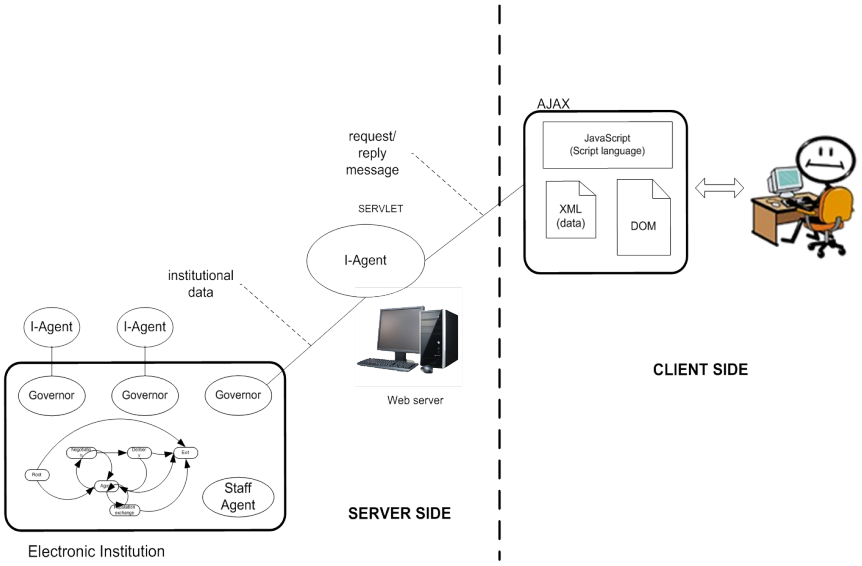


Fig. 2. Architecture of the Charter Management System, Charms

The central element in the link between the user and the EI is the servlet (or web server), which is activated once the user (client side), using a web browser, establishes the connection with the server side. At that point in time, the I-

Agent tries to enter the EI, if successful, the EI informs the server and client sides about the current status of the I-Agent inside the EI (see Figure 3). Note that while the I-Agent is seen as a servlet from the point of view of the web server, it is seen as a normal agent from the point of view of the EI.

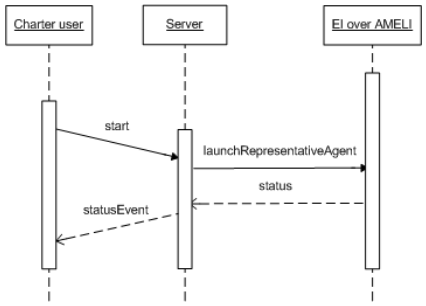


Fig. 3. Sequence diagram for the start request sent by the user

The client receives XML events from the servlet describing the changes produced in the EI that update the state of the charter's flow. These changes are shown to the user, and can be of four types:

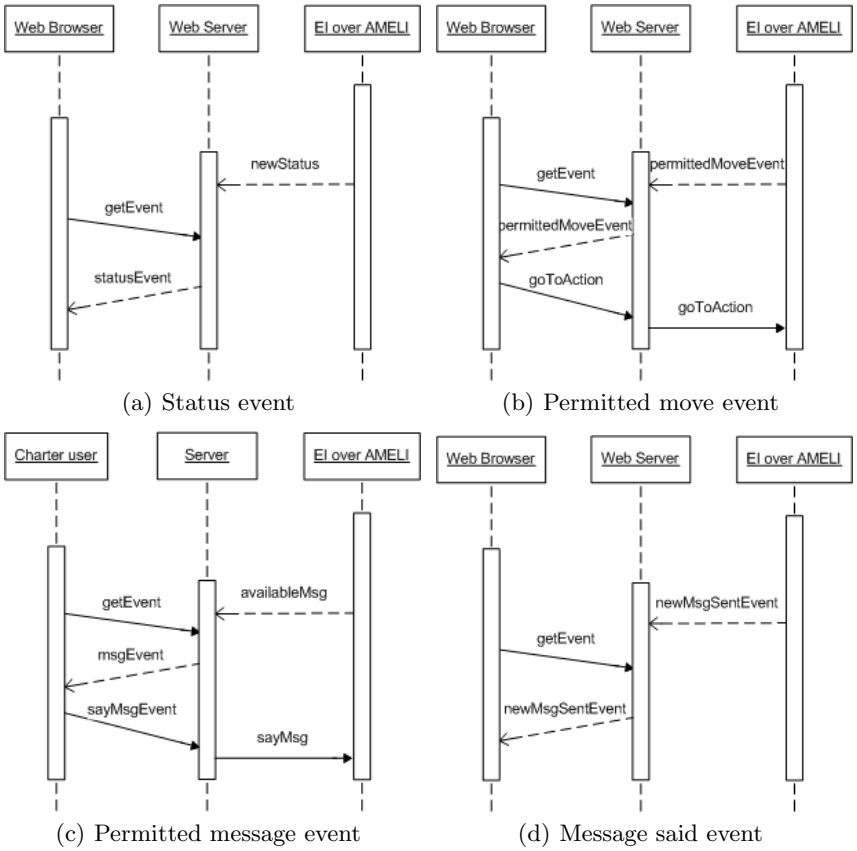
- Status event: Informs a user about her position in the charter's flow.
- Permitted move event: Informs a user where she can move forward in the charter's flow.
- Permitted message event: Informs the user the illocutions she can say.
- New message said event: Informs the user of the illocutions addressing her.

Simultaneously, the I-Agent, acting as a servlet, receives events in XML format from the client application (which runs locally in the user's computer as a web application). We refer to this kind of events as actions because they represent the actions that charter users want to perform in the charter. The I-Agent then acts accordingly, inside the EI, as soon as it receives an action from the user. These actions can be of two types:

- Say message action: Informs the corresponding I-Agent about the illocution the user wants to say in the EI.
- Go to action: Informs the corresponding I-Agent where the user wants to go from her current position in the general charter's flow.

So far we have focused on the back-end issues of Charms. We have described how the different elements of the Charms architecture exchange events among them. But how are these events processed and displayed in the web-based interface that final users see? It should be first said that the current implementation of Charms does not allow to customize what users see. In other words, we use a

fixed web template to build the web-based interface. This template is mainly divided into four tables, each one including one of the following information: where the user is in the charter's flow (state), received messages, permitted actions and the messages the user can send (see Figure 7 for an example). Depending on the type of the event received, the client side displays it in the corresponding table. The AJAX technology allows the modification of only one section of the screen that is concerned with the new information.



(a) Status event (b) Permitted move event

(c) Permitted message event (d) Message said event

Fig. 4. Sequence diagrams for the events occurring inside the EI that are relevant to the charm's user: new status, permitted movement, permitted and sent messages

The visualization process involves the I-Agents and the client application. As aforementioned, the link between the client side and the EI is the I-Agent (or servlet). The I-Agent stores the list of events coming from the EI that need to be notified to the user. The client application asks the I-Agent for new events and displays them properly in the web browser. To do that, the client application implements a cycle timer expressed in milliseconds. With each cycle, the client

application requests new events from the I-Agent (see Figure 4). When the I-Agent receives a request for events from the client, it pulls events from the list and builds a response for the client, including the new events in XML format. In Figure 4 we present the UML diagrams of the relations between events and actions involving the charter user, I-Agent and EI. Dashed and solid arrows represent events and actions, respectively.

6 Interdisciplines, a motivating example

Interdisciplines (`http://www.interdisciplines.org/`) is a web-based platform aiming at organizing online interdisciplinary meetings such as workshops, seminars and conferences. It addresses interdisciplinary researches in philosophy, cognitive science and social science. Since its creation in 2001, the Interdisciplines website has organised sixteen meetings with more than 150 papers. The site has more than 4000 registered users who participate playing the roles: *administrator*, *author*, *forum moderator*, *panelist* and *reader*.

Interdisciplines helps meeting organisers to create and run interdisciplinary meetings, which differ from standard academic meetings. In standard academic meetings researchers submit papers to be considered for presentation. Usually, every submitted paper is reviewed by members of a program committee, who evaluate the manuscript and determine if it is accepted for the meeting. By contrast, the organisers of an Interdisciplines meeting invite a set of authors to submit papers (with a maximum of 3000 words) to the meeting, and the papers are then debated within a discussion forum with the assistance of a scientific committee. We note that authors may also invite other researchers to coauthor papers with them. That request, however, needs to be approved by either an administrator or a moderator of the meeting. Furthermore, moderators and panelists assess papers according to several criteria, such as focus, originality, arrangement, methodology and writing. These reviews are not public and can only be accessed by the reviewers who made them, the authors of the manuscripts and the meeting administrators.

Meeting organisers must determine when and for how long a paper is debated. Generally, that period of time goes from 15 to 30 days. For every discussion thread, the meeting organisers also define who will play the roles of moderator and panelist. The debate of a contribution starts out when a panelist or reader makes a comment on the paper. The authors may respond to comments made by the panelists. Messages from panelists and authors are automatically shown in the forum threads that the website holds. However, comments made by users playing the role of reader must be validated first by the moderators.

7 Implementing Interdisciplines in Charms

In this section, we use Charms to implement Interdisciplines. In fact, the implemented version is a slightly lighter version of Interdisciplines, which, for example,

does not take into consideration the different sections of a paper. This lighter version is used to keep the EI model clear and simple for the reader. As mentioned in Section 3, the modelling task requires the definition of roles, performative structure and interaction protocols. The roles, as introduced by the previous section, are: that agents can play in Interdisciplines are: *admin* (or the conference organiser), *moderator*, *panelist*, *author* (the main author), *coauthor*, and *reader* (or the guest that may read the website).

In addition to the above, EI requires one more role: $staff$. Except for staff, the roles above are played by human users. The staff role belongs to the EI and takes care of different aspects related to the good functioning of the system. Hence, the institution designers are in charge of defining and implementing the actions the staff role will do. As we mentioned in Section 3, this can be done by using aBUILDER, the agent development tool provided by EIDE.

The performative structure of an EI defines how agents are permitted to move among the different scenes and transitions. Figure 5 shows the performative structure of the proposed EI for Interdisciplines. In this figure, the blocks represent the different scenes, and the arcs linking these scenes represent the permitted transitions between scenes. Note that transitions are labelled with the roles that are allowed to perform this transition. For example, only an agent playing the role "admin" is allowed to move from the "Admission" scene to the "CreateConf" scene, where conferences are created. Also note that agents enter and leave the EI through the "begin" and "end" scenes, respectively. In summary, Figure 5 states that agents playing non-institutional roles must authenticate in the *Admission* scene before doing anything inside the EI. Afterwards, administrators can ask the staff agent for creating a conference in the *CreateConf* scene. This request includes the list of parameters we have mentioned in Section 6 needed to create an Interdisciplines conference. In the *Submission* scene authors can submit papers to the conference, invite other researchers to coauthor papers and propose text about their papers. Coauthors, however, can only submit papers and propose text about their papers but not inviting other researchers to participate in their papers. Except for admin, all the roles are involved in the *RunConf* scene, where they participate in paper discussions. Next we provide further details of the interaction protocols for the Admission, CreateConf, Submission and RunConf scenes:

Admission: Every agent must be authenticated by the staff agent in this scene. The staff agent determines who can enter the system by validating user authentication data (LoginData), which consists of an email and a password.
CreateConf: This interaction protocol allows conference organisers to create new conferences. Two roles may participate in this scene: staff and admin. The admin agent (meeting organiser) requests the staff to create a new meeting. The request includes the list of papers, moderators and panelists as well as relevant dates for the conference.
Submission: Basically this interaction protocol allows researchers to submit papers to the conference. Roles staff, admin, author, and coauthor are involved in this process. Authors can invite other researchers to co-write

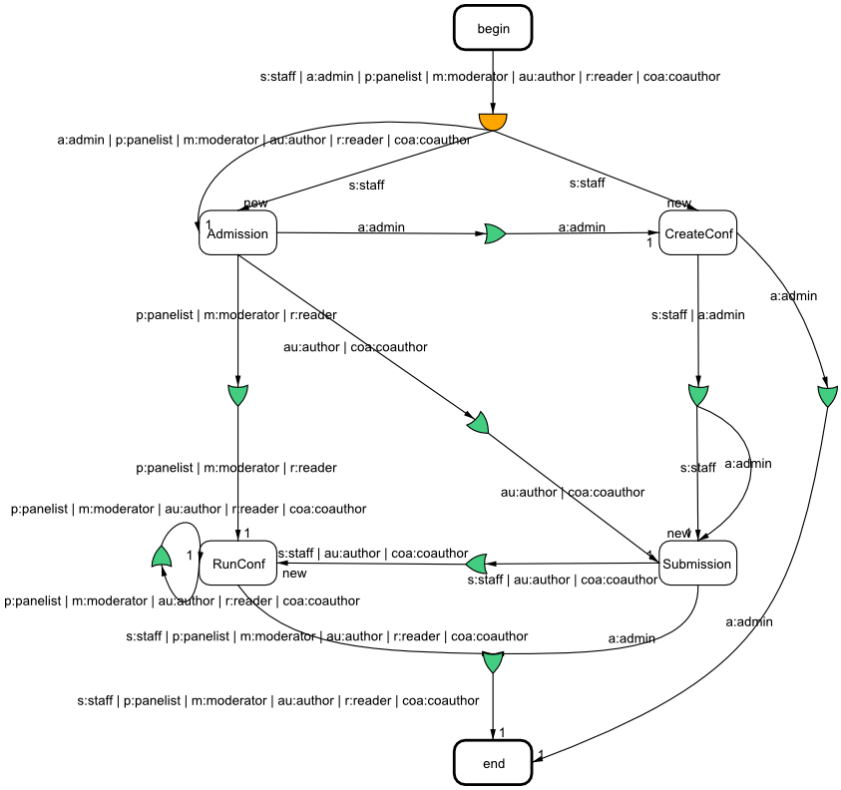


Fig. 5. Interdisciplines' performative structure

papers. However, these requests have to be validated by any administrator agent. The new invited researchers play the coauthor role. Coauthors can submit a paper to the conference but cannot invite new researchers to participate in the writing of the paper. Authors and coauthors of a given paper can share pieces of texts in order to improve the paper.

RunConf: This is the place where papers are discussed. Moderators, panelists, authors, coauthors and readers are the agents permitted to participate in this scene. They make comments on papers by exchanging messages. Panelists and readers react to papers, authors and coauthors reply and moderators validate messages from readers. Note that while comments coming from authors, coauthors and panelists are automatically published, those coming from readers need to be reviewed by the moderators.

As an example we present in Figure 6 and Table 1 the interaction protocol and permitted illocutions for the submission scene, respectively. The scene starts at its initial state *w0* where authors and coauthors can exchange proposal of the papers between themselves (arrow #13 in Figure 6), submit their papers (arrow #1), or inform the administrator about their desire to invite new researchers

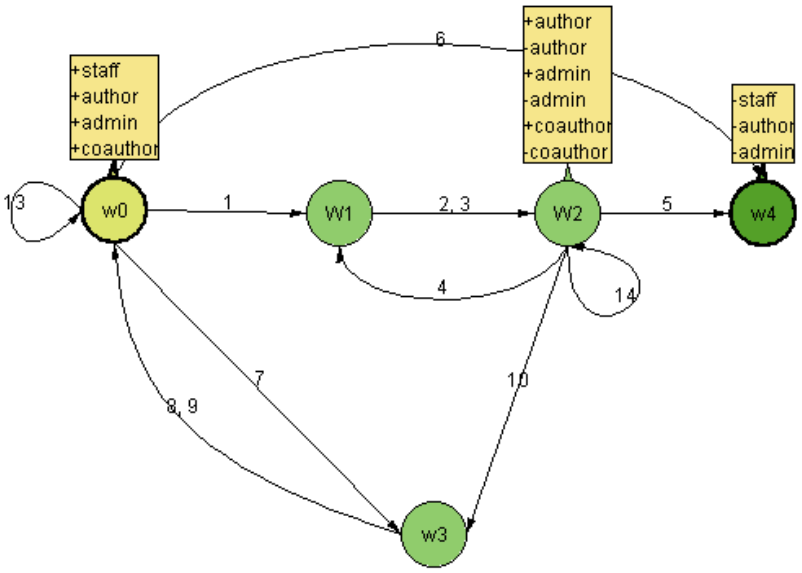


Fig. 6. Submission scene protocol specification

arrow	**illocution**	**meaning**
1, 4	(submit (?coa coauthor) (?a admin) (?pa:Paper))	*coa* informs *a* that she wants to submit paper *pa*.
2	(success (!a admin) (!coa coauthor) (?pa:Paper))	*a* says *coa* paper *pa* was successfully submitted.
3	(failure (!a admin) (!coa coauthor) (?pa:Paper))	*a* says *coa* submission of the paper *pa* fails.
7, 10	(inviteCoauthor (?au author) (?a admin) (p:Person))	*au* asks *a* to invite person *p* to cowrite a paper.
13, 14	(proposeVersion (?coa coauthor) (all authors) (pa:Paper))	*au* proposes a new version for paper *m* to authors.
8, 11	(accepted (!a admin) (!au author) (p:Person))	*a* informs *au* that person *p* was accepted as coauthor.
9, 12	(denied (!a admin) (!au author) (p:Person))	*a* informs *au* that person *p* was denied as coauthor.
5, 6	(inform (!s staff) (all) (close))	staff informs the others about the end of the scene.

Table 1. Permitted illocutions for the Submission scene.

to cowrite papers with (arrow #7). The first keeps the scene at the same state. The second action moves the scene to state *w1*. And the third action moves the scene to state *w3*. At state *w3*, the administrator can accept or deny coauthor requests (arrows #8 and #9). At state *w1*, the administrator automatically checks whether the submission process has succeeded or failed, and informs the

author or coauthor who submitted the paper of the result (arrows #2 and #3). This moves the scene to state *w2*. The actions permitted at state *w2* are similar to those permitted at state *w0*: authors and coauthors can exchange proposal of the papers (arrow #14, which keeps the scene at the same state), submit their papers (arrow #4, which moves the scene to state *w1* again), and authors can inform the administrator of their desire to invite new researchers to cowrite papers (arrow #10, which moves the scene to state *w3*). The main difference between state *w0* and *w2* is that some agents are allowed to exit the scene at this state (-author, -admin, -coauthor). Finally, we note that a staff agent informs the rest of the agents when the submission scene ends (arrow #5 at state *w0* and arrow #6 at state *w2*). All agents leave the scene at stage *w4*.

7.1 Web-based user interface for Interdisciplines

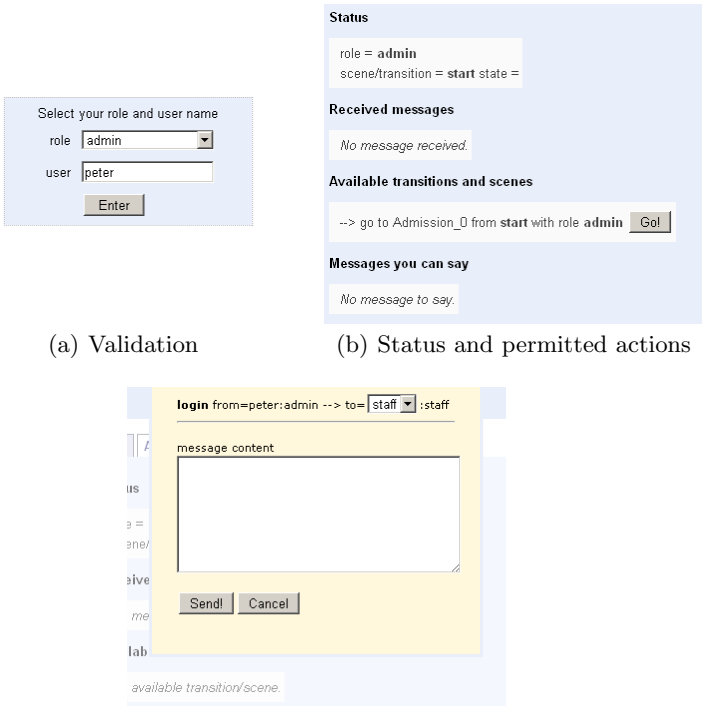
In the previous section, we illustrated how Interdisciplines may be specified through an EI. We now move on to discuss the automated UI of Charms.

The first step a human user has to do is to connect to the URL with the UI, in our case: http://localhost:8080/charms-webapp/. As we can see in Figure 7(a), the user is required to choose the name and the role she wants to play in the charter. In our running example, the user **Peter**, playing the role **admin**, is at the transition **start**. In general, when the user submits the name and role to start playing in a given charter, Charms forwards the user to a new web page (Figure 7(b)), which contains the following:

Status: This described the user's role and its location w.r.t. the step flow.
Received messages: The messages the user received since she entered the scene.
Permitted actions: The actions the user may perform according to its role and status in the charter's flow. These include:
Moves: The moves the user is allowed to do, i.e. the list of transitions or scenes the user may go to.
Messages The messages the user is allowed to say.

The charter specifies that Peter can only move to the **Admission** scene after entering the system. Figure 7(b) illustrates that Peter neither has received any message from other users nor is he permitted to say any messages, and the only transition it may choose is to go to the **Admission** scene.

Afterwards, Peter goes into the Admission scene by pressing the **Go!** button of Figure 7(b). Once Peter is in the scene, he can only authenticate into the system by sending a message to the internal staff agent. By clicking on the **Message content** button, a pop-up appears as we see in Figure 7(c). It contains a description of the message, the possible recipients of it and a blank text area, where the user should fill out the content of the message, in this case an e-mail and a password as was defined in the specification of the EI.



(a) Validation (b) Status and permitted actions

(c) Message content

Fig. 7. Sample of the web-based user interface for Interdisciplines

7.2 Changing the Interdisciplines charter

In this section we show how easy it is to make a change in the above charter and how this change is automatically reflected in the web user interface. For the sake of simplicity, the change is quite simple with the aim to illustrate the potential of the approach.

As illustrated by Figure 8, we note that amongst the roles are the author role and its sub-role, the coauthor. This means that all the actions that are allowed to a coauthor, are allowed also to an author, but not vice versa. In what follows, we will focus on the actions that these roles has the right to perform.

In the original charter specification both, authors and coauthors, are allowed to submit a paper as shown by arrows #1 and #4 in the Submission scene protocol (see Figure 6). According to the charter, authors could invite a coauthor, submit a full paper and propose content for a paper (i.e. text); whereas coauthors could only submit and propose text. This is automatically reflected in the web interface that the users playing the role of author and coauthor see when they are in the submission scene, as shown for the coauthor case in Figure 10(a).

Now we want to change this basic functionality and modify the charter to limit the actions that a coauthor is allowed to do. In this new version, only au-

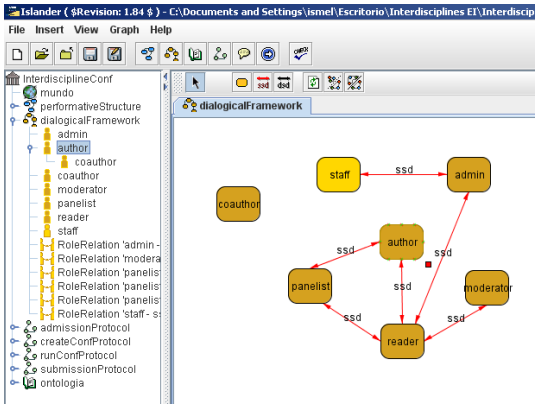
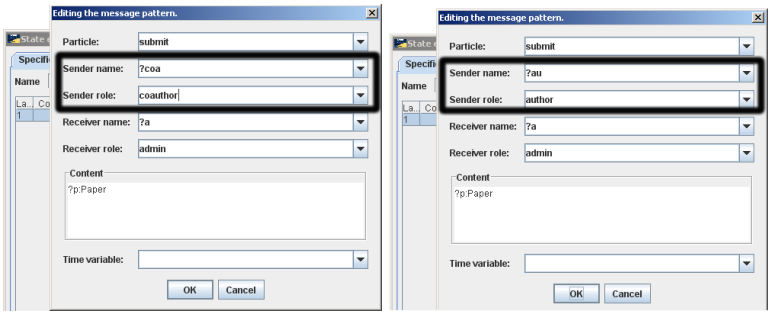


Fig. 8. Roles structure in Interdisciplines

thors are permitted to submit a paper whereas coauthors are limited to propose new texts. Table 2 summarizes the actions allowed to authors and coauthors in the original charter (column Before) and the functionality we want for the new version (column After).

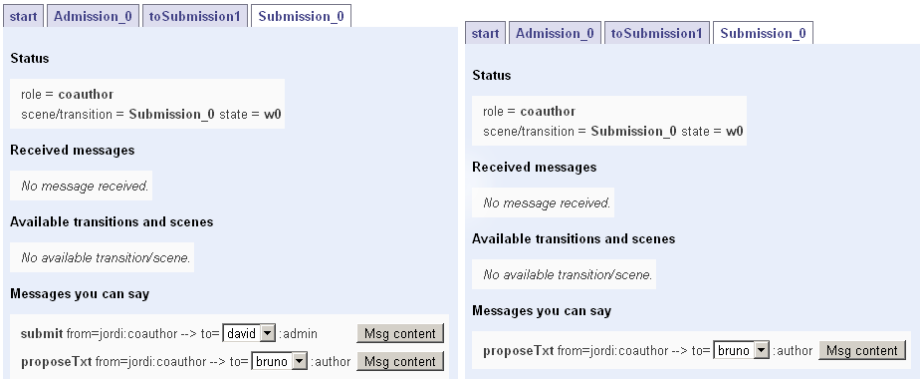
In this case, updating the charter is quite easy. In the specification protocol of the original charter, the role associated with arrows #1 and #4 is "coauthor" (see Figure 9(a)). This means that only the agents playing the role of coauthor and author can submit a paper (remember that coauthor is a sub-role of author). If we want that only authors are allowed to submit a paper what we have to do is to change the role associated with arrows #1 and #4 so that only agents playing the role author can perform that action (see Figure 9(b)).

The key aspect of the approach is that once the charter specification has changed, no further modifications are necessary in the web interface. Now, when a coauthor arrives to the Submission scene she sees the screen in Figure 10(b) where the functionality of submitting a paper has automatically disappeared.



(a) Before the change (b) After the change

Fig. 9. Submission scene protocol specification



(a) Before the change (b) After the change

Fig. 10. Submission scene web interface for a coauthor

Actions	Author		Coauthor	
	Before	After	Before	After
Invite coauthor	X	X	-	-
Submit	X	X	X	-
Propose text	X	X	X	X

Table 2. Permitted actions for authors and coauthors before and after the change

8 Using Charms

In this section we enlist the set of requirements needed for installing and executing Charms. We also provide guidelines on how to deploy and test new charters specifications with Charms. The first thing to do is to download and install Charms. The latest version of Charms is available at:
`http://eru.iiia.csic.es/export/ftp/pub/charms/charms.zip`.

The software has been designed and tested in systems with the following characteristics:

Hardware requirements: Any computer with a minimum capacity and network connection will do the job.

Software requirements:

- Operative Systems: Windows XP or later/Fedora Core release 5 (Bordeaux).

- Apache HTTP Server 2.2.0 or later (with the port 8080 accessible)

- Java(TM) Development Kit, version 6 [7]
- Apache Ant 1.7.0 or later [8]
- Apache Tomcat 6.x [9]

To install and test Charms the next steps have to be followed:

- **Step 1** Decompress charms.zip in any folder.
- **Step 2** Edit `/CHARMS/etc/project.properties` file and set `tomcat.home` variable to the proper path in your computer.
- **Step 3** Edit `/CHARMS/www/charms-webapp/WEB-INF/web.xml` file and set the server name where Charms is located, by default `localhost`.
- **Step 4** Specify a charter with the ISLANDER editor for EIs [10] (for further details see [Esteva et al., 2002]).
- **Step 5** Add the new charter specification to `/CHARMS/etc/` directory.
- **Step 6** Edit `/CHARMS/etc/charms/ei.conf` file and set `specification` tag to proper path of the new charter specification added in Step 3.

You should consider the steps 4, 5 and 6 only if you want to test your own charter specification. Otherwise, Charms will use the Interdisciplines charter specification described in Section 7. At that point, you are ready to execute your charter. To do that, perform the following actions:

1. Open a system console and locate yourself in `/CHARMS/etc/`.
2. Execute `rmiregistry`.
3. Execute `ant charms charms-webapp-deploy`.
4. Execute `ant tomcat-start`.

Now you should be able, by using a browser (we recommend Firefox although any browser should work), to open the website `http://server-host:8080/charms-webapp` (where server-host is the name of the server you have set in the step 3). This website allow users to participate remotely into the charter you have specified in the step 4.

9 Conclusions and future work

This paper introduced using electronic institutions as a mechanism to specify and execute charters. We presented Charms, a running system that takes advantage of all the tools that are already available for the specification and execution of electronic institutions. The novelty of our work is that the system permits the

[7] Available at `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
[8] Available at `http://ant.apache.org/bindownload.cgi`
[9] Available at `http://tomcat.apache.org/download-55.cgi`
[10] Available with the EIDE at `http://e-institutions.iiia.csic.es`

maintenance of a web based user interface that is automatically generated from a charter specification. The changes in the specification are reflected in the web user interface in a fully automated manner, and without the need of any manual modification to the UI. We note that available technologies, such as Java Server Faces, may be used in aiding the design. However, Charms differs from these technologies in that it generates the mapping between the electronic institutions and the user interface in an automated manner. To our knowledge, there is not such a thing implemented yet in electronic institutions.

With this tool, managing charters becomes a simple task. Defining different policies, norms, or social behaviour for particular components in a complex structure is done in a modular and automatic way. The behaviour allowed to agents (or human users) is automatically and transparently derived from these charters. Thus, when managing pieces of knowledge, the user is offered the functionality that is appropriate to each type of knowledge in a dynamic way. That is, as knowledge evolves by the actions of others, the user's allowed functions are updated in front of his/her eyes. The feeling of being in a dynamic world is then perceived in a very vivid way.

Currently, the generated web user interface is very simple and, although it works as a proof of concept, it is not friendly enough for a final user. The visual aspect and the functionality of the interface has to improve. The current interface refers to low level electronic institution structures, using terms like scenes, transitions, roles, etc. These low-level details should be hidden. To solve that, a mapping between low level electronic institution elements and domain dependent elements is necessary. This mapping should be part of the specification.

Acknowledgement

This work has been supported by: the LiquidPublications project (`project.LiquidPub.org`), funded by the Seventh Framework Programme for Research of the European Commission under FET-Open grant number 213360; the Agreement Technologies project (`www.agreement-technologies.org`), funded by CONSOLIDER CSD 2007-0022, INGENIO 2010; and the CBIT project on community-building information technology, funded by the Spanish Ministry of Science and Innovation under grant number TIN2010-16306.

References

[Boile et al., 2006] Boile, J., Cardella, M., Blanyalet, S., Juric, M., Carey, S., P., C., Coene, Y., Geminiuc, K., and Zirn, M. (2006). *BPEL Cookbook*. Packt Publishing.

[Cardelli and Gordon, 1998] Cardelli, L. and Gordon, A. (1998). Mobile ambients. In *Proceedings of the First international Conference on Foundations of Software Science and Computation Structure*, volume 1378 of *LNCS*, pages 140–155. Springer-Verlag.

[Esteva et al., 2002] Esteva, M., de la Cruz, D., and Sierra, C. (2002). Islander: an electronic institutions editor. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1045–1052, New York, NY, USA. ACM.

[Esteva et al., 2004] Esteva, M., Rosell, B., Rodriguez-Aguilar, J. A., and Arcos, J. L. (2004). Ameli: An agent-based middleware for electronic institutions. *Autonomous Agents and Multiagent Systems, International Joint Conference on*, 1:236–243.

[Harel and Politi, 1998] Harel, D. and Politi, M. (1998). *Modeling Reactive Systems with Statecharts: The Statemate Approach*. McGraw-Hill, Inc., New York, NY, USA, 1st edition.

[Jennings et al., 1996] Jennings, N. R., Faratin, P., Johnson, M. J., and Wieg, M. E. (1996). Using intelligent agents to manage business processes. In *In Proceedings of the First International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 345–360.

[Milner, 1999] Milner, R. (1999). *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press.

[Shepherdson et al., 1999] Shepherdson, J. W., Thompson, S. G., and Odgers, B. (1999). Cross organisational workflow co-ordinated by software agents. In *Proceedings of the Workshop on Cross-Organisational Workflow Management and Co-ordination*.

[Sierra et al., 2004] Sierra, C., Rodriguez-Aguilar, J. A., Noriega, P., Esteva, M., and Arcos, J. L. (2004). Engineering multi-agent systems as electronic institutions. *UPGRADE The European Journal for the Informatics Professional*, V(4):33–39.

[Sierra et al., 2007] Sierra, C., Thangarajah, J., Padgham, L., and Winikoff, M. (2007). Designing institutional multi-agent systems. *LNCS*, 4405:84–103.