Infrastructures to Engineer Open Agent Environments by Means of Electronic Institutions

Dave de Jonge, Juan A. Rodriguez-Aguilar, Bruno Rosell i Gui, Carles Sierra

Artificial Intelligence Research Institute, IIIA
Spanish National Research Council, CSIC
08193 Bellaterra, Spain
{davedejonge,jar,rosell,sierra}@iiia.csic.es

Abstract. Electronic institutions provide a computational analogue of human institutions to engineer open environments in which agents can interact in an autonomous way while complying with the norms of an institution. The purpose of this paper is twofold. On the one hand, we lightly survey our research on coordination infrastructures for electronic institutions in the last ten years. On the other hand, we highlight the research challenges in environment engineering that we have tackled during this journey as well as promising research paths for future research on the engineering of open environments for multi-agent systems.

1 Introduction

As the complexity of actual-world applications increases, particularly with the advent of the Internet, there is a need to incorporate organisational abstractions into computing systems that ease their design, development, and maintenance. Electronic Institutions (EIs) are at the heart of this approach [3]. Just like any human institution, an EI is a place where participants come together and interact according to some pre-defined norms. An EI warrants that the norms of the institution are enforced upon its participants, and thus prevents them from misbehaving. Therefore, an EI provides the *environment* in which agents can interact in an autonomous way within the norms of the institution.

From a computational point of view, an EI realises an environment for agents in the sense proposed in [22], namely it provides "the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources". Considering the levels of support, as defined in [22], which an environment may provide, EIs focus on two particular levels of support: (i) an abstraction level that shields agents from the low-level details of deployment, and (ii) mediated interaction between agents. With this aim, EIs provide a computational infrastructure for agent environment design, as discussed in [21], and the mechanisms to enforce and monitor the norms and laws that apply to a multi-agent system in a given environment. More precisely, an EI is implemented itself as a multi-agent system composed of two types of (internal) agents, the so-called staff agents and governor agents. While governor agents mediate the interactions of (external) agents within the environment, staff agents manage the norms in the institution. The dynamics of the environment is restricted to those external agents that satisfy the social laws represented by the norms and enacted by the coordinated actions of governors and staff agents. With this aim, governors and staff agents employ synchronisation mechanisms for interaction mediation, along the lines of [17].

EIs have been under development for almost 20 years, which has resulted in a large framework consisting of tools for implementing, testing, running and visualizing them [3]. Therefore, research on EIs has strongly focused on contributing with infrastructures and tools that support systematic engineering support of environments for multi-agent systems, which has been identified as one of the challenges in environment engineering [20].

Since their inception, EIs have focused on computationally realising environments for multi-agent systems. The purpose of this paper is to show the evolution of the infrastructures that we have employed over these years to engineer *open* agent environments by means of electronic institutions. We aim at illustrating the way the surveyed infrastructures deal with different degrees of openness.

With this aim, we survey the four generations of coordination infrastructures for EIs developed so far. For each generation, we dissect: (i) the features of the coordination infrastructure running an EI; (ii) the facilities provided to support human participation; and (iii) a case study illustrating some application built with the aid of EIs. Furthermore, after introducing each generation of the corresponding EI infrastructure, we provide a critical analysis that motivates the need to move from one generation to the next one.

First, we lightly touch upon first generation (1G) EIs, whose tremendous development complexity and effort motivated the subsequent introduction of AMELI [13], a general-purpose coordination infrastructure for second generation (2G) EIs. 2G EIs represented a very significant advance with respect to 1G EIs because they provided MAS engineers with well-founded tools that significantly eased development. Thus, the development of 2G EIs consists of two main stages: the specification of institutional rules by means of ISLANDER [12], a graphical specification tool; and their subsequent execution by AMELI, a general-purpose coordination infrastructure. Hence, instead of programming an EI, as it was the case with 1G EIs, 2G EIs allow a MAS engineer to focus on *specifying* the rules of an EI.

Nonetheless, some drawbacks hinder the applicability of 2G EIs. On the one hand, 2G EIs cannot be designed and enacted on-line. Consider, for instance, a business scenario where manufacturers are allowed to meet on-line and arrange collaborations on the fly to produce the goods requested by customers. These companies do not know beforehand neither what customers will request, nor the partners to collaborate with, nor the rules of their collaborations. In this setting, if a group of manufacturers aim at employing an EI to structure their collaboration, this must be designed and enacted on-line upon request. On the other hand, human participation in 2G EIs is highly complex, since it forces users to learn about the inner formal languages employed by EIs.

Third-generation (3G) EIs were conceived to support the on-line enactment of EIs. This facility is provided in the realm of the Agreement Technologies Environment (ATE) [1], an OSGi-compliant [2], open environment that provides the seamless interplay of agents and services. An EI is offered as an organisation service within the ATE. Thus, during the run-time operation of the ATE, agents participating in the environment are allowed to start and run an EI at any given time. In other words, EIs can be started on-line, during the run-time operation of the ATE, depending on agents' needs. And yet, notice that 3G EIs do not offer any facilities for the on-line design of EIs, though this is an important requirement for applications requiring the on-line enactment of collaborations (e.g. crowdsourcing, supply chain formation).

Indeed, some open environments may require that communities of participants design electronic institutions on-line. The fourth generation of tools for EIs is completely web-based, the tools allow users to specify EIs using a graphical editor that is much simpler than ISLANDER, and less expressive as well. There are no free lunches. Specifications can include web services and agents. Users can search specifications and launch them, participate in them through a web browser and enjoy an automatically generated web interface. The programming effort has been reduced enormously in this generation of tools. Finally 4G EIs are implemented as a P2P network of nodes that allows to exploit the benefits inherent to P2P systems (e.g. self-organisation, resilience to faults and attacks, low barrier to deployment, privacy management, etc.).

The paper is organised as follows. Sections 2, 3, 4, and 5 review the features of the first, second, third, and fourth generations of EIs. Section 6 further analyses the relationship between EIs and environments, and Section 7 draws some conclusions and sets paths to future research.

2 First generation: Electronic institutions enacted by ad-hoc coordination infrastructures

First-generation (1G) EIs are represented by the early developments reported in [10] and [18]. Each of those EIs implemented an electronic auction house inspired on the fish market described in [16] with different focuses. First, [10] is a proof-of-concept, PVM-based implementation of the fish market. The use of PVM[15] was intended to support and ease the distributed execution of agents and communication protocols. As reported in [18] (see section 5), further prototype development of the fish market followed using PVM and MPI for inter-networking and C and EU-Lisp. All of these implementations focused on low-level communication aspects and shared the commonality of being rather costly. Finally, the implementation in [18], thanks to the use of the Java language, allowed us to focus on modularity and search for the core objects required to implement structured interactions in an EI.

All in all, the EIs implemented from [10] to [18] focused on developing each their own, ad-hoc coordination infrastructure. Furthermore, in all cases, support for human interaction was provided by means of tailored graphical user interfaces. However, notice that all of them realised open, distributed environments that supported the remote participation of trading agents.

3 Second generation: Electronic institutions enacted by a general-purpose coordination infrastructure

The experience during the implementation of the above-described first-generation electronic institutions taught us that the design and development of open MAS is a highly complex task. This motivated research on tools that ease MAS engineers' development effort. Thus, in [4], we introduced EIDE, an integrated development environment for electronic institutions that supports all the stages of the design and development of an open MAS as an electronic institution. EIDE clearly differentiates two main stages: the specification of institutional rules; and their subsequent execution by a coordination infrastructure. In this way, instead of programming an electronic institution, as it was the case for first-generation electronic institutions, a MAS engineer can focus on *specifying* the rules of an electronic institution.

In this section we describe the key features of AMELI, the coordination infrastructure provided by EIDE to run electronic institutions. Moreover, we also comment on the tools provided by EIDE to support human interaction. Finally, we analyse the drawbacks of second-generation electronic institutions.

3.1 AMELI: a core infrastructure for electronic institutions

The infrastructure (i.e. set of institutional agents) that enables the execution of EIs is called AMELI [13]. AMELI enables agents to act in an electronic institution and controls their behaviour. Its main functionalities are:

– to provide a way for different agents with different architectures to communicate with one another without any assumption about their respective internal architectures; and
– to enforce a protocol of behaviour as specified in an institution specification upon the agents. This means that AMELI makes sure that the agents can only do those actions that the protocol allows them to do.

AMELI was conceived as 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, a graphical editor for EI specifications. 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 institutional 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 enforce their consequences.

AMELI is composed of three layers: a communication layer, which enables agents to exchange messages, a layer composed of the external agents that participate in an EI, and in between a social layer, which controls the behaviour of the participating agents. The social layer is implemented as a multi-agent system whose institutional agents are responsible for guaranteeing the correct execution of an EI according to the specification of its rules.

The participation of each agent in an EI through AMELI is handled by a special type of institutional mediator, the so-called governor. An agent must be able to communicate with its governor, but this only requires that the agent is capable of opening a communication channel. Since no further architectural constraints are imposed on external agents, we can regard AMELI as agent-architecture neutral.

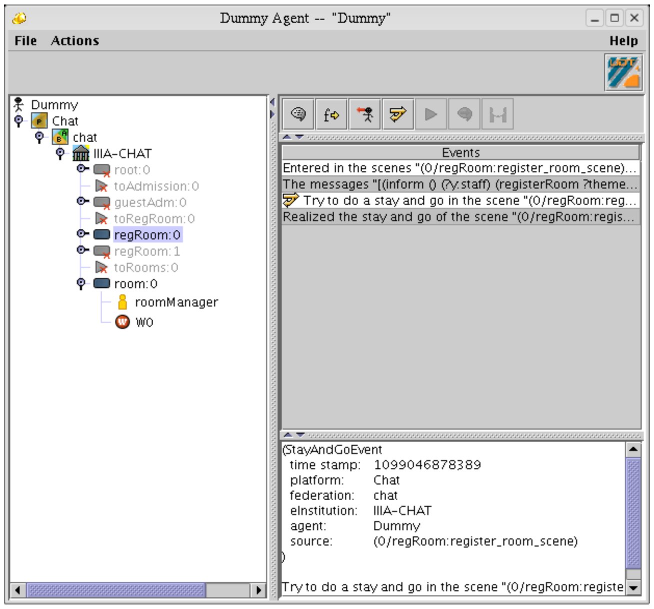
The current implementation of AMELI can either use JADE (Bellifemine et al., 2001) or a publish-subscribe event model as communication layer. When employing JADE, the execution of AMELI can be readily distributed among different machines, permitting the scalability of the infrastructure. From the point of view of the participating agents in an EI, AMELI is communication neutral, since they are not affected by changes in the communication layer.

3.2 Human participation

Instead of providing a domain-dependent graphical user interface for human interaction as first-generation electronic institutions did, EIDE provides a general-purpose interface, the so-called *dummy agent*. Figure 1 depicts a screenshot of the dummy agent as displayed to a user. On the left-hand side, the interface shows a tree-like structure that displays the run-time structure of an electronic institution together with the agents participating in the institution. Such run-time structure is composed of the on-going activities (scenes) in an electronic institution. On the right hand side, the figure shows the events, including the speech acts exchanged between agents, occurring within a particular scene selected by the user. Notice that the dummy agent offers a user the possibility of building speech acts to interact with the rest of agents within an institution. In other words, humans interact using the same language employed by software agents. Therefore, in order to participate in an electronic institution by means of a dummy agent, a user must know: (i) how an electronic institution is formally represented; and (ii) the formal language employed by EIDE to represent speech acts.

3.3 Case study

EIDE was employed in the development of an actual-world application: the Multi-Agent System for FIsh Trading (MASFIT) [9]. MASFIT allows buyers to remotely participate in several wholesale fish auctions simultaneously with



```
Entered in the scenes "(0/regRoom:register_room_scene)...
The messages "[(inform () (?y:staff) (registerRoom ?theme...
Try to do a stay and go in the scene "(0/regRoom:reg...
Realized the stay and go of the scene "(0/regRoom:regis...
```

```
(StayAndGoEvent
  time stamp:    1099046878389
  platform:      Chat
  federation:    chat
  eInstitution:  IIIA-CHAT
  agent:         Dummy
  source:        (0/regRoom:register_room_scene)
)

Try to do a stay and go in the scene "(0/regRoom:registe
```

Fig. 1: Dummy agent user interface.

the help of software agents, while maintaining the operation of traditional auctions. The participation of buyer agents in actual-world auctions is mediated by an electronic institution run by AMELI. MASFIT's institution controls buyers' access to the auctions, provides them with information, and collects their bids during the auctions. To permit this, the actual auction systems running at different auction houses were connected to the developed institution. MASFIT interconnects multiple auction houses, and thus provides structure to a federation of auction houses. MASFIT guarantees equal conditions for both human buyers physically participating at the auction and software buying agents participating through the Internet.

3.4 Analysis

2G EIs represent a significant advance with respect to 1G EIs. Very importantly, notice that the tools developed to design and run 2G EIs are general-purpose. That means that the very same tools can be employed to design and run different EIs. In this new context, MAS engineers do not have to cope any longer with low-level implementations of communication and coordination protocols, as it was the case for 1G EIs. Instead, they can only concentrate on *specifying* the rules for EIs. Therefore, 2G EIs allow to shift the development effort from programming to specifying.

Nonetheless, several drawbacks may hinder the applicability of 2G EIs. First, notice that the design of the rules of the institution is carried out off-line by a single user, the institution designer. That means that 2G EIs cannot be employed in application domains that require the on-line enactment of an EI. Second, AMELI enacts open environments whose rules are static, and hence cannot change over time. Third, human interaction is highly intricate, since it forces human users to learn the inner formal languages of EIs. This is too heavy a burden for human users, and clearly motivates the need for more user-friendly EI interfaces.

4 Third generation: Electronic institutions

Third generation (3G) EIs were conceived to allow agents the on-line enactment of EIs. To support such functionality, we developed the so-called Agreement Technologies Environment (ATE) [1], a service-oriented, open environment that provides the seamless interplay of agents and services. Then, an EI is offered as an organisation service within the ATE. Thus, during the run-time operation of the ATE, agents participating in the environment are allowed to start and run an EI at any given time. In other words, EIs can be started during the run-time operation of ATE depending on agents' needs.

Next, in section 4.1 we describe the implementation of Agreement Technologies Environment. As part of this description, we detail how EIs are offered within the environment as organisation services. Thereafter, in section 4.2 we describe a case study that shows how EIs are designed and enacted on-line, namely during the operation of the ATE. Finally, section 4.3 explains how 3G EIs simplify human participation with respect to 2G EIs, while section 4.4 provides a critical analysis.

4.1 Agreement Technologies environment

The Agreement Technologies Environment (ATE) [1], formerly introduced in [1], is an environment that provides the seamless interplay of agents and services. We chose to implement the ATE as a service-oriented environment based on the OSGi [2] technological framework.

The OSGi technology is a *de facto* industry standard that defines a dynamic component system for Java. OSGi reduces complexity by providing a modular architecture for today's large-scale distributed systems as well as for small, embedded applications. Therefore, the OSGi programming model is called to fulfill the promise of component-based systems and beyond, since OSGi is intended to allow an application to emerge from dynamically assembling a set of components.

Modularity is at the core of the OSGi specifications and embodied in the *bundle* concept. Bundles are the OSGi components built by developers. AN OSGi bundle is like a Java JAR file, but it hides everything unless explicitly exported. The OSGi architecture offers a service model whose aim is to have bundles collaborate. The services layer connects bundles in a dynamic way. This is feasible because a bundle can register a service, it can get a service (from another bundle), and it can listen for a service (from another bundle) to appear or disappear. Furthermore, services are dynamic: bundles can register and withdraw their services at any time.

To summarise, OSGi provides a very powerful architecture to support the dynamic collaboration of services offered by bundles. We developed the ATE to be OSGi-compliant. However, in order to turn it into an environment were also agents were able to participate, it was endowed with bundles providing three types of agent support services: (i) agent-to-service interactions; and (ii) agent-to-agent interactions; (iii) multi-agent collaborations. Figure 2 shows the architecture of the ATE developed on top of OSGi. The agent support services mentioned above correspond to the following groups of services (bundles):

Service tools. They allow agents, as well as services, to discover and call remote services in a distributed environment.

Agreement services. They are intended to help agents to reach agreements (by means of argumentations), and monitor and manage agreements (by means of the use of trust and reputation ratings provided by the trust/reputation service, which in turn may require the use of the ontology service).

Organisation services. They provide services to enact collaborations between agents. For instance, an EI is offered as a service. In fact, notice that all the services provided by EIDE, the development environment for 2G EIs, are provided as organisation services.

Figure 2 also shows environment and user interface services. On the one hand, environment services ensure that dependencies between modules are met and facilitate collaborations in a distributed environment. On the other hand, user interface services provide interfaces to the ATE and define an interface to help ATE services to implement interactions with human users.

As mentioned above, EIs are offered as organisation services. In the example that follows, we illustrate the on-line design and enactment of EIs by groups of agents taking part in the ATE.

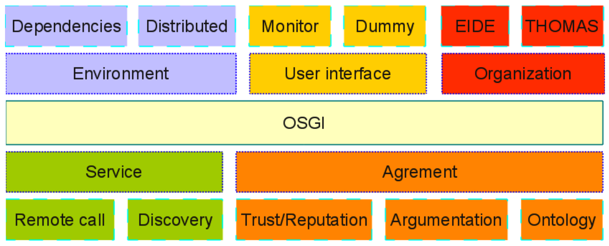


Fig. 2: ATE services and tools.

4.2 Case study

In [1] we describe the *Assembling Business Collaborations for Multi-agent System* (ABC4MAS) platform, a collaboration environment to support the rapid assembly of agent-oriented business collaborations. ABC4MAS allows: (i) to set up a collaboration environment as a virtual organization; (ii) to reach agreements within the collaboration environment to form short-term business collaborations to manufacture customer-requested goods; (iii) to enact business collaborations to produce goods; and (iv) to track the performance of agents within business collaborations to build their trust and reputation. ABC4MAS was built as an application running on the Agreement Technologies Environment described in section 4.1 above.

Figure 3 shows the architecture of the ABC4MAS platform. First, the platform counts on a collaboration environment enacted as a virtual organisation thanks to the THOMAS Platform [5]. When a customer within this collaboration environment issues an order, a new virtual organisation is spawned to service the request. This virtual organisation contains all business parties that may take part in the production of the requested good together with an auctioneer. The auctioneer is in charge of assessing an optimal supply chain by means of a mixed auction [8], an auction protocol aimed at solving supply chain formation problems. Hence, after clearing the auction, the auctioneer obtains the specification of a supply chain that it translates into an EI specification. Thereafter, the auctioneer launches an EI by means of the AMELI service, which is fed with the EI specification generated by the auctioneer. AMELI then tracks each and every action as defined in the supply chain, allowing the auctioneer to monitor: (1) which entity is performing each task, (2) that all the agreements are being fulfilled, and (3) that no task is overdue. When AMELI detects that the production process has finished, both the EI and the virtual organization started to serve the customer request are terminated. At production time, during the execution of an EI, there is a service, the Supplier Relationships Management (SRM) service [14] is in charge of collecting information about the agents taking place in the supply chain to keep up to date the trust and reputation values of the agents participating in the collaboration environment.

4.3 Human participation

In section 3.2 we argued that the interaction of a user with a dummy agent is intricate. 3G EIs try to overcome this problem. Thus, they offer a methodology, the so-called *human interaction within hybrid environments regulated through electronic institutions* (HIHEREI) [6], to help an institution designer produce a more user-friendly interface to electronic institutions. Unlike a dummy agent, which was conceived as a general purpose interface to any electronic institution, an interface produced by means of HIHEREI is tailored to the domain for which the electronic institution is designed. Therefore, HIHEREI interfaces are domain-specific.

4.4 Analysis

Although 3G EIs allow to dynamically start EIs as services, they do not offer any facilities to support the on-line design of EIs, neither by a single designer

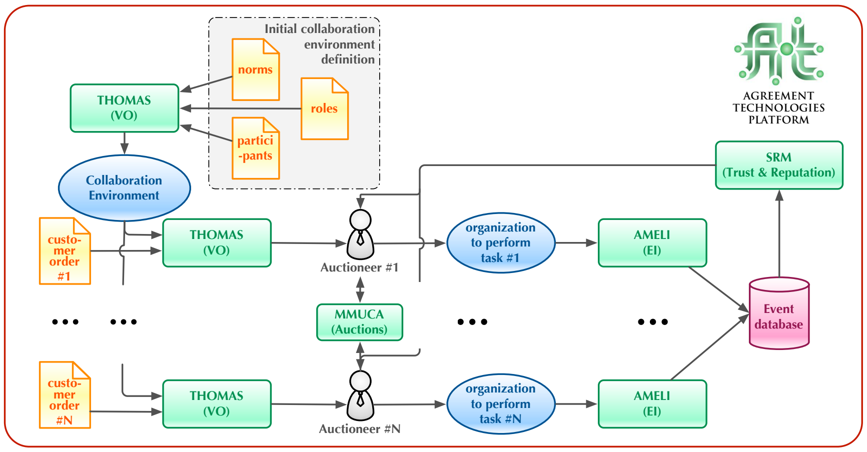

Initial collaboration environment definition
norms
roles
partici-pants
THOMAS (VO)
Collaboration Environment
custo-mer order #1
custo-mer order #N
Auctioneer #1
Auctioneer #N
MMUCA (Auctions)
organization to perform task #1
organization to perform task #N
AMELI (EI)
SRM (Trust & Reputation)
Event database
AGREEMENT TECHNOLOGIES PLATFORM

Fig. 3: ABC4MAS platform architecture.

nor by a group of designers. And yet, this feature is particularly interesting in some application domains. For instance, consider a social network that lets its users form their own communities and choose and enact their own rules. Furthermore, although 3G EIs ease human participation by means of the HIHEREI methodology, this new approach still suffers from some drawbacks. First, it is now the designer, and not the end user, the one who must know about the formal languages used by EIs. Furthermore, HIHEREI does offer a methodology, but the designer is responsible for programming users' interfaces. In other words, HIHEREI shifts the burden from the end user, as it is the case in 2G EIs, to the EI designer. Second, the interface produced by HIHEREI is as static as the dummy agent offered by 2G EIs. More precisely, such an interface starts out with a single EI specification that cannot change at run time. Consider again a social network like the one mentioned above. In that setting, the interface of a user may need to change depending on the different communities where the user participates. In the next section we show how fourth generation EIs have successfully tackled these issues.

5 Fourth-generation: Peer-to-peer electronic institutions

In a recent document by IBM, "Device Democracy: Saving the future of the Internet of Things"[1] a case is made about the uncertain future of centralised approaches in the context of networks composed of billions of interconnected devices. Centralised approaches would become prohibitively expensive, would not protect privacy and would not make business models endure.

It is our ambition that electronic institutions become a pervasive mechanism to co-ordinate very large networks of humans and devices and thus a centralised

[1] ibm.biz/devicedemocracy

approach seems a bad solution for the future. We thus wanted to recover the spirit of the implementations of the first generation using libraries like PVM or MDI but with modern technologies.

Peer-to-peer (P2P) networks appear as the natural option to implement distributed electronic institutions nowadays as they provide a number of desirable properties. They are very robust in that there is no single point of failure. If a node fails, the other nodes may continue the computation, in many cases without any loss or with minimal loss of information. P2P networks scale very well as a new node increases the overall amount of computation requests but it also brings in resources in the form of e.g. memory or CPU cycles. Nodes are equally privileged and the quality of service they receive does not depend on whether they can afford expensive cloud servers.

In the last decade many applications and technologies have been built using this approach: file sharing (e.g. Gnutella or BitTorrent), distributed storage (e.g. Symform, Freenet, Yvi), or web search engines (e.g. Yaci, Faroo). P2P platforms (i.e. set of interconnected nodes over TCP/IP protocols) provide different features: trust, authentication, data persistency, state guarantee, anonymity, etc.

In this section we will describe an implementation of a P2P software node, named PeerFlow, that extends AMELI (of the second generation) with a number of P2P features. Thus, the fourth generation grows out of the second generation incorporating an improved human interface over HIHEREI from the third generation and leaving out the Agreement Technologies toolbox structuring the third generation. We may however incorporate some of these tools in the near future.

5.1 PeerFlow Basic infrastructure

We have built PeerFlow on top of the *Freepastry*[2] library. A free and open-source Java library that implements Peer-to-Peer network [11, 7, 19]. Freepastry provides a number of useful features such as the routing of messages, or the possibility to create broadcast messages.

5.2 P2P electronic institution internal agents

A running institution is managed by a number of agents called *scene managers* and *governors*. On a Peer-to-peer system one needs to decide on which node in the network each of these agents will be running. For this reason we have added another type of agent to the framework called the *device manager*. Each node in the network runs exactly one device manager. Whenever a new agent needs to be launched, the device managers determine where that agent is going to be launched. In the current implementation this is decided randomly, but in future implementations the device managers will apply negotiation to make such decisions, taking into account the capacity of each node (e.g. bandwidth and CPU power).

[2] http://www.freepastry.org/FreePastry/

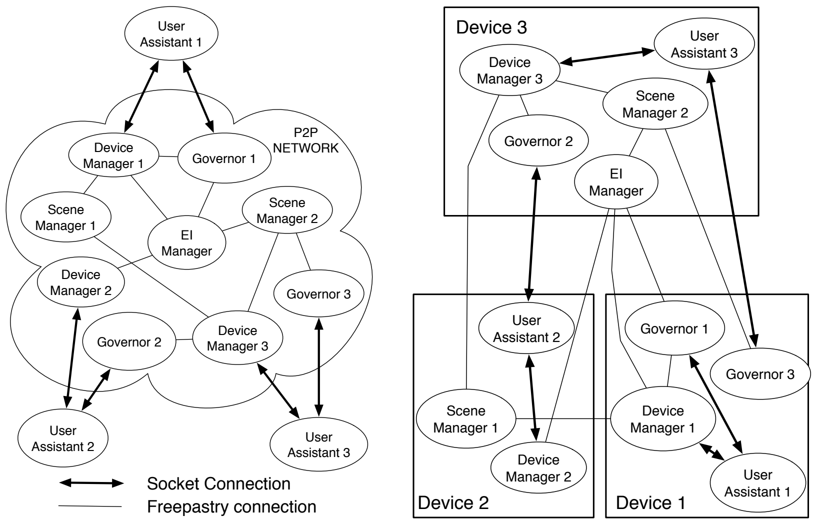


Fig. 4: Diagram of the Peer-to-Peer Electronic Institutions topology

Note that the agents participating in the EI are not directly inside the P2P network. Instead, they are connected to a governor through a direct socket connection, which is inside the P2P network. We have chosen this model for security reasons, because messages in a P2P network do not always go straight from sender to receiver, but may make several 'hops' between nodes of the network before arriving at the receiver. This means that agents participating in the institution would be able to intercept those messages and manipulate them.

5.3 Peerflow distributed database

Peerflow also provides a distributed database where users can publish their EI-specifications, search for existing specifications, and search for running instances of electronic institutions. This is implemented using the indexer/search library *Apache Lucene*.[3] Each node in the network has its own repository which is maintained by the device manager. When a query to the database is made, this query is sent to all the device managers in the network, and each of them sends back a reply, if possible.

5.4 Building software agents. What's different?

Programming an agent that interacts in a P2P EI can be done in two ways. One way consists of extending an existing Java agent that abstracts away all the underlying communication protocols. The other way is to make use of a *rest governor*. The intended actions of the agent are then sent as http requests to

[3] http://lucene.apache.org/

the rest governor. The advantage of the second method is that one can use any kind of programming language or technology that allows making web requests, but has the disadvantage that one has to deal with the http protocol on a lower level.

5.5 Human participation

Earlier versions of EIDE, in the second and third generations, where intended as a framework for software agents and did not provide significant support for humans to participate in Electronic Institutions. For many purposes, however, it is desirable to have both humans and agents participating. One can for example imagine an auction house in which bids are made by humans, but in which the tasks of the auction house, such as registration of participants and leading the auction are taken care of by automated agents. Therefore, we have extended the existing EI framework with a Graphic User Interface (GUI) that allows humans to enter into an EI and interact with other users or software agents. This new extension is called GENUINE, which stands for GENerated User INterface for Electronic institutions

We think that this extension could be especially useful for the development of a new type of social network, where users can set up sub-communities, each with its own rules and protocols. We think that the fact that many social networks are nowadays existing next to each other is inefficient and is due to the fact that users are not able to adapt norms and protocols to their own needs. Electronic Institutions would provide a solution to this problem. Another advantage of allowing human users to interact in an EI is that it enables developers of institutions to test an institution during its development, without having to program any agents. While the EI is under development human users can take the place of the software agents that would later participate in it, for testing purposes. This can highly increase the speed of development.

Our tool automatically generates a default user interface based on the EI specification, without the need for extra programming. But, on the other hand, if one does require a more case-specific user interface, it still provides an API that enables any web designer to easily design a custom GUI without the need for much knowledge of Electronic Institutions or Java programming. Our approach is completely web-based, meaning that the GUI is in fact a website, implemented using standard web-technologies such as HTML5, Javascript and AJAX. In short, we have developed our framework with the following goals:

- To allow people to interact in an EI through a web browser.
- To have a generic GUI that is generated automatically.
- To allow any web designer to easily design a new GUI, if desired.
- To allow testing of an EI under development, before having implemented its agents.

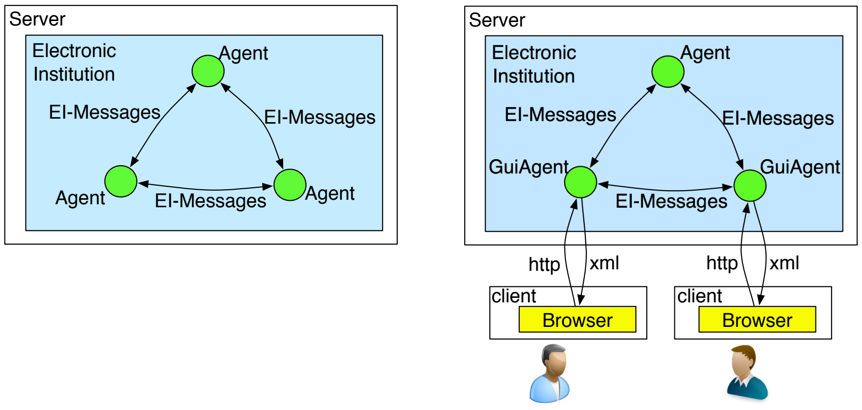


Fig. 5: Left: a 'classic' EI with only software agents. Right: an EI with one software agent and two users. In these images the agents are located on a server, but they may just as well run on a P2P node.

Our Framework A human user would interact in an EI by clicking buttons in a browser window. To allow these actions to have effect in the EI, we have implemented a software agent that represents the user inside the EI and that executes the actions requested by the user. This agent is called the *GuiAgent*. Its current implementation does not do anything autonomously, but, if necessary, it can be extended with more sophisticated capabilities, such as giving intelligent strategic advice to the user.

When developing the framework we took into account that, on one hand, one may want to have a good-looking GUI that is specifically designed for a given institution. But, on the other hand, one may not want to develop an entirely new GUI for every new institution, or one may want to have a generic GUI available to test a new EI during its development, so that one can postpone the design of its final GUI until the EI is finished. Therefore, our framework allows for both. It generates a GUI automatically from the EI-specification, but at the same time provides an API that enables web designers to easily create a custom GUI for every new EI. The framework is used on top of the existing EI-framework and consists of the following components:

– A Java agent called *GuiAgent* that represents the user in the EI.
– A Java component that encodes all relevant information the agent has about the current state of the institution into an xml file.
– A Javascript library called *GenuineConnection* that translates the xml file into a Javascript object called *EiStateInfo*.
– A Javascript library called *GenuineDefaultGUI* that generates a default Graphic User Interface (as html) based on the EiStateInfo object.

How it Works In order for a user to participate in an institution, there must be an instance of that EI running on a server or in the P2P network. To join the institution, the user then needs to open a web browser and navigate to the institution's URL. The process then continues as follows:

1. A web page including the two Javascript libraries is loaded into the browser.
2. When the page is loaded, GenuineConnection library sends a login request to the server or to the P2P network.
3. Upon receiving this request the server or network starts a GuiAgent for the user and, depending on the specific institution, other agents necessary to run the institution.
4. When the GuiAgent is instantiated it analyzes the EI-specification to retrieve all static information about the institution.
5. The GenuineConnection library starts a polling service that periodically (typically several times per second) requests a status update from the GuiAgent.
6. When the GuiAgent receives a status update request it asks its Governor for the dynamic information about the current status of the institution.
7. The GuiAgent converts both the static and the dynamic information into xml which is sent back to the browser.
8. The GenuineDefaultGUI Javascript library then uses this information to update the user interface (more information about this below).
9. The user can now execute actions in the institution or move between its scenes by clicking buttons on the web page.
10. For each action the user makes, the GenuineConnection library sends a HTTP-request to the GuiAgent.
11. The GuiAgent uses the information from the http-request to create an EI-message which is sent like any other message in a standard EI.

As explained, the GuiAgent uses two sources of information: static information from the EI-specification stored on the hard disk of the server and dynamic information from the Governor. The static information consists of:

- The names and protocols of the scenes defined in the institution.
- The roles defined in the institution.
- The ontology of the institution.

While the dynamic information consists of:

- The current scene and its current state.
- The actions the user can take in the current state of the scene.
- For each of these actions: the parameters to be filled out by the user.
- Which agents are present in the current scene
- Whether it is allowed to leave the scene and, if yes, to which other scenes the user can move.

Generating the GUI Every time the browser receives information from the GuiAgent, it updates the GUI. This takes place in two steps, respectively handled by the two Javascript libraries. In the first step the GenuineConnection library converts the received XML into a Javascript object called EiStateInfo, which is composed of smaller objects that represent the static and dynamic information as explained above.

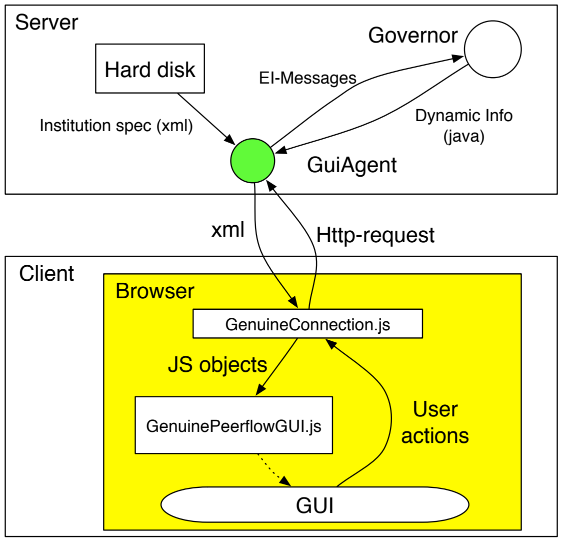


Fig. 6: The components necessary to generate the GUI. Solid arrows indicate exchange of information. The dashed arrow indicates that the GUI is created by the GenuineDefaultGUI library

In the second step the EiStateInfo-object is used by the GenuineDefaultGUI library to draw the GUI. This GUI is completely generic, so it looks the same for every institution. If one requires a more fancy user interface tailored to one specific EI, one can write a new library that replaces the GenuineDefaultGUI.

The fact that these two steps are handled by two different libraries enables designers to reuse the GenuineConnection when designing a new GUI, without having to worry about how to communicate with the EI and retrieve the necessary information from the EI to draw the GUI. All information will be readily available in the EiStateInfo-object, so one only needs to determine how to display it on the screen.

Customizing the GUI A customized GUI-generator can retrieve all necessary information from the EiStateInfo-object. For example: if the user chooses to make a bid in an auction, the GUI-generator would read from the EiStateInfo-object that an Integer parameter must be set to represent the price the user wants to bid. The programmer of the GUI-generator should make sure that whenever a parameter of type Integer is required, the GUI displays an input-control that allows the user to introduce an integer value.

The fact that one can also define user-defined types in an EI adds a lot of flexibility. Suppose for example that one would like a user to record an audio file and send this in a message to another agent. Electronic Institutions do not support audio files by default. However, the institution designer may define a new type with the name 'Audio'. Once the user chooses to send a message that includes audio, the EiStateInfo-object will indicate that a parameter of type Audio is needed. A customized GUI-generator could then be implemented such that a microphone is activated whenever this type of parameter is required.

5.6 Case study: PRAISE

We have used the P2P infrastructure in an application to music learning. As one-to-one teaching is very expensive we have built electronic institutions that support the interactions of communities of learners. We call each institution a 'lesson plan' where each scene in the institution represents a musical activity of students and teachers (upload songs, analyse progress, give feedback, etc.), see figure 7. In this use case, human interaction is facilitated by GENUINE, you can see an example of interface from the point of view of student Carles in Figure 8 and the result of an automatic audio analysis in Figure 9. For further details go to www.iiia.csic.es/praise.

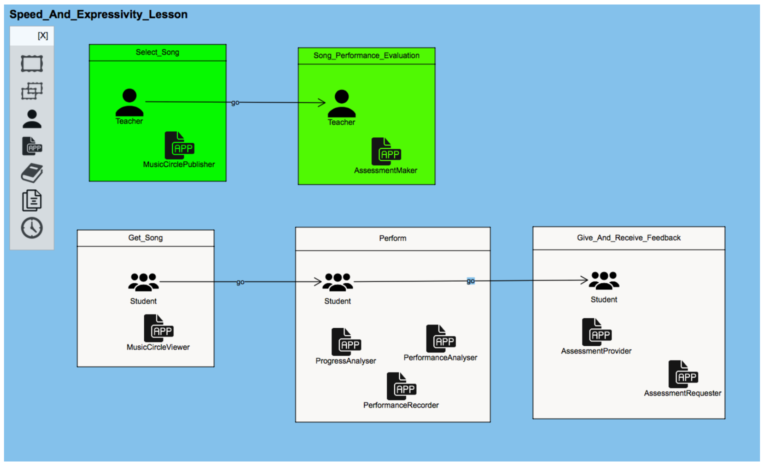
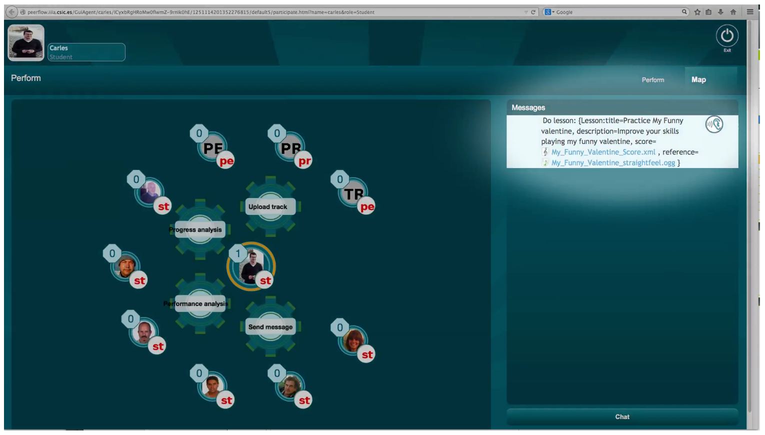


Fig. 7: An example of a lesson plan. Boxes represent scenes, arrows flow of participants and APPs represent web services.


peerflow.iiia.csic.es/GuiAgent/carles/ICyxbRgHRoMw0flwmZ-9rnik0hE/1251114201352276815/default5/participate.html?name=carles&role=Student
Carles
Student
Exit
Perform
Perform
Map
Messages
PE
pe
PR
pr
TR
pe
st
Upload track
Progress analysis
Performance analysis
Send message
Chat

Do lesson: {Lesson:title=Practice My Funny valentine, description=Improve your skills playing my funny valentine, score= My_Funny_Valentine_Score.xml , reference= My_Funny_Valentine_straightfeel.ogg }

Fig. 8: A group of students within an activity to check their progress using automatic analysis tools.

5.7 Case study: WeCurate

WeCurate is an image browser for collaboratively curating a virtual exhibition from a cultural image archive —group curation. WeCurate allows users to synchronously view media and enables negotiation about which images should be added to the group's image collection. Further, it accelerates the navigation of extensive museum databases and provides a platform for sociocultural experiences, combining the actions of autonomic agents and users to facilitate decision-making. WeCurate is tasked with establishing the users' presence in the shared experience by enabling communication around the deconstruction and appropriateness of the media, representing social proxies, and making agent and group members' actions manifest to everyone. In figures 10, 11, and 12 you can see the ad-hoc interface developed over P2P for WeCurate. For further details visit http://www.iiia.csic.es/ace/project.

6 On the relationships between electronic institutions and environments

It is time to analyse the relationship between EIs and environments by considering how research on EIs has contributed to the challenges posed by research on environments for multi-agent systems. As mentioned in 1, since their inception, EIs have focused on computationally realising environments for multi-agent systems. With this aim, research on EIs has contributed with infrastructures for agent environment design [21], such as illustrated by the different infrastructures surveyed in this paper. EIs propose a particular software architecture for

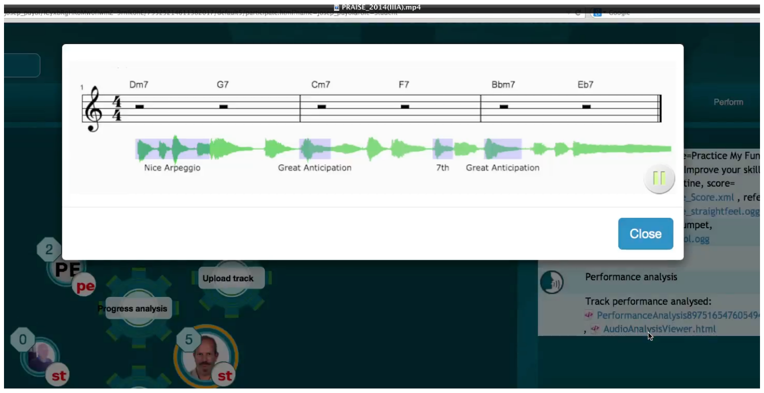


Fig. 9: An example of automatic feedback.

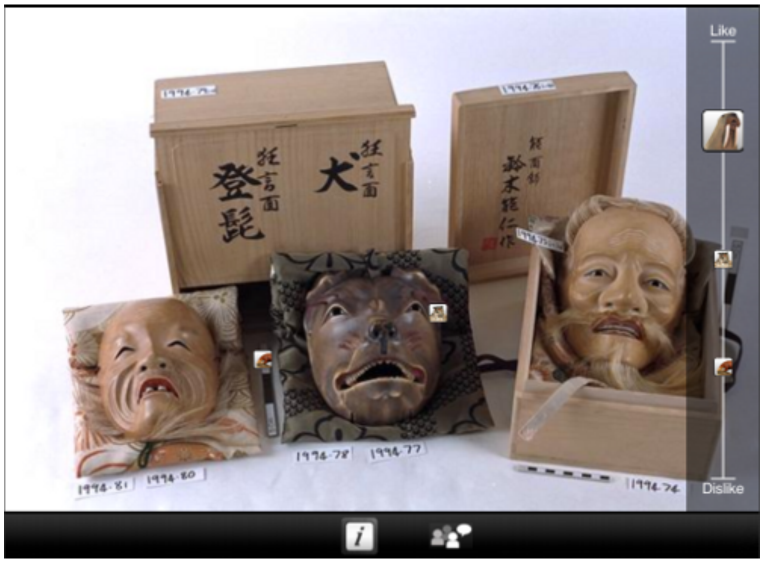


Fig. 10: The selection scene, which aims to gauge the interest level of users in the proposed image.

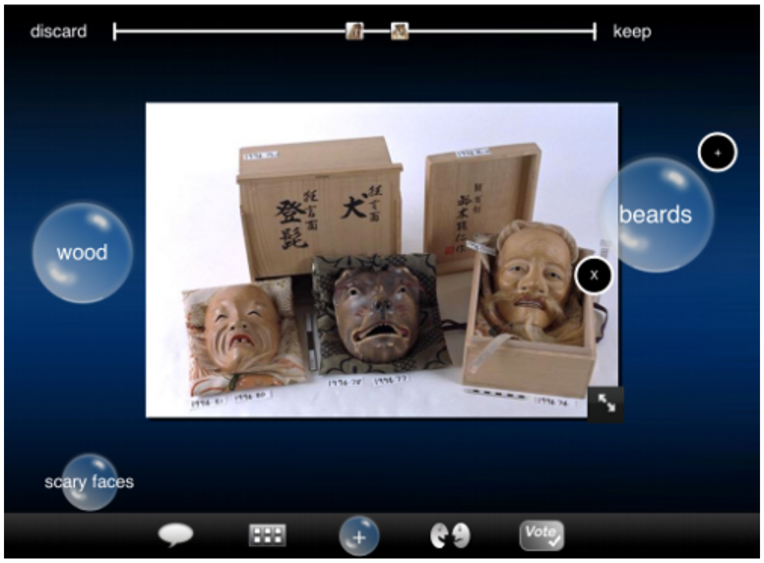


Fig. 11: The forum scene, which allows users to engage in a discussion about the image, once it has been deemed interesting in the selection scene.

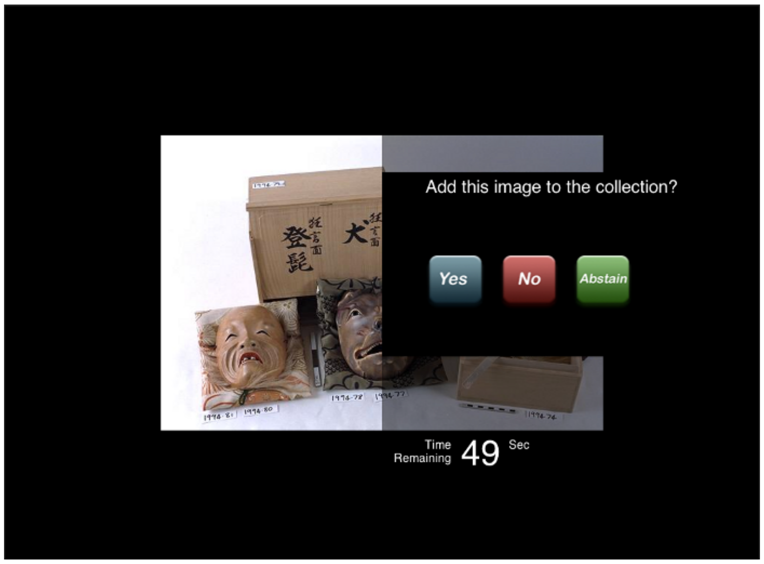


Fig. 12: The vote scene, where users can vote on whether they wish to store the image to their collection.

an agent environment [22] that distinguishes between the agents participating in the environment and the institutional agents (staff agents and governor agents) in charge of guaranteeing that the norms of the institution are satisfied. Thus, institutional agents in EIs are in charge of the responsibilities of an agent environment as detailed in [22] and they share the required synchronisation mechanisms to support social interactions according to norms.

Notice though, that research on EIs has not only focused on infrastructures for agent environment design. In a more general sense, research on EIs has strongly focused on tools that support the systematic engineering of agent environments, identified as one of the main challenges in environment engineering in [20].

The infrastructures surveyed in this paper have tried to illustrate how EIs can help enact environments. Thus, 1G and 2G EIs help enact environments that are open to agent participation but whose rules are static. Next, 3G EIs allow to enact agent environments at run time. Moreover, EIs are enacted in the realm of the Agreement Technologies Environment (ATE). That means that an environment, ATE, can host agent environments that run as EIs. Finally, 4G EIs go even one step further, since they allow agents to *design* and *enact* their own environments at run-time. Notice therefore that 3G EIs and 4G EIs share a commonality: there is an environment that allow agents to start their own environments. This makes us revisit the three levels of environment support discussed in [22]. We argue that it is worth considering a fourth level of support: *interaction design level*. This is the level of support offered to agents so that they can design themselves their own interactions, namely their own environments within the environment they are immersed in.

7 Conclusions

In this paper we have shown the evolution of the coordination infrastructures that we have employed for almost 20 years to engineer open environments as EIs. First, we have discussed that the first evolution of our coordination infrastructures was motivated by the need for supporting the development of EIs. Thereafter, our focus on new application domains (e.g. social networks, social computing, collaborative on-line learning) posed new challenging requirements that pushed the development of the next generation of coordination infrastructures for EIs. In short, there have been two main drivers guiding the evolution of EIs, namely:

– The need for designing and running *social environments*. We have pursued to increase openness to computationally realise environments where users can dynamically create and enact their own interaction environments. On the one hand, users are allowed to collaboratively design on-line the rules of their own interaction environments. On the other hand, the coordination infrastructure is ready to enact the decentralised design and execution of such interaction environments.

– The need to facilitate human interaction. The next decade will witness an increased amount of mixed societies where humans, devices, sensors and actuators will constitute an Internet of Things as the substrate of new applications like smart homes, service robotics or ambient intelligence. The simplification of the human interaction with co-ordination infrastructures like EIs is key for those applications.

These two needs have by no means been fully satisfied. Thus, future generations of coordination infrastructures for EIs will definitely focus on making progress along these two directions.

Acknowledgements

This paper has been partially funded by the following projects: TIN2012-38876-C02-01, PRAISE (FP7-318770), CollectiveMind (TEC2013-49430-EXP), ACE (Autonomic software engineering for online Cultural Experiences), and the Generalitat of Catalunya (2014 SGR 118).

References

1. Toni Penya Alba, Boris Mikhaylov, Marc Pujol-Gonzalez, Bruno Rosell, Jesús Cerquides, Juan A. Rodríguez-Aguilar, Marc Esteva, Angela Fabregues, Jordi Madrenas, Carles Sierra, Carlos Carrascosa, Vicente Julian, Mario Rodrigo, and Matteo Vassirani. *An Environment to Build and Track Agent-Based Business Collaboration*, volume 8, chapter 36, pages 611–624. Springer, 2013.
2. OSGi Alliance. Osgi. http://www.osgi.org, cited February 2015.
3. Josep Lluís Arcos, Marc Esteva, Pablo Noriega, Juan A. Rodríguez-Aguilar, and Carles Sierra. Engineering open environments with electronic institutions. *Eng. Appl. of AI*, 18(2):191–204, 2005.
4. Josep Lluis Arcos, Marc Esteva, Pablo Noriega, Juan Antonio Rodríguez-Aguilar, and Carles Sierra. An integrated development environment for electronic institutions. In *Software agent-based applications, platforms and development kits*, pages 121–142. Birkhäuser Basel, 2005.
5. E. Argente, V. Botti, C. Carrascosa, A. Giret, V. Julian, and M. Rebollo. An abstract architecture for virtual organizations: The THOMAS approach. *Knowledge and Information Systems*, 29(2):379–403, October 2010.
6. Ismel Brito, Isaac Pinyol, Daniel Villatoro, and Jordi Sabater-Mir. Hiherei: Human interaction within hybrid environments. pages 1417–1418, Budapest, Hungary, 2009.
7. Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 140–145. ACM, 2002.
8. Jesús Cerquides, U. Endriss, Andrea Giovannucci, and Juan A. Rodríguez-Aguilar. Bidding languages and winner determination for mixed multi-unit combinatorial auctions. pages 1221–1226, 2007.
9. Guifre Cuni, Marc Esteva, Pere Garcia, Eloi Puertas, Carles Sierra, and Teresa Solchaga. Masfit: Multi-agent system for fish trading. *ECAI*, 16:710, 2004.

10. C Di Napoli, C Sierra, M Giordano, P Norlega, and MM Furnari. A pvm implementation of the fishmarket multiagent system. In *ISAI/IFIS 1996. Mexico-USA Collaboration in Intelligent Systems Technologies. Proceedings*, pages 68–76. IEEE, 1996.
11. P Draschel and A Rowstron. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, pages 329–350, 2001.
12. Marc Esteva, David de la Cruz, and Carles Sierra. ISLANDER: an electronic institutions editor. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, pages 1045–1052, 2002.
13. Marc Esteva, Bruno Rosell, Juan A. Rodríguez-Aguilar, and Josep Lluís Arcos. Ameli: An agent-based middleware for electronic institutions. In *Proceedings of AAMAS*, pages 236–243, 2004.
14. Angela Fabregues and Jordi Madrenas-Ciurana. Srm: a tool for supplier performance. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1375–1376. International Foundation for Autonomous Agents and Multiagent Systems, 2009.
15. Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine—A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, Mass., 1994.
16. Pablo Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor. PhD thesis Universitat Autònoma de Barcelona, 1997.* Number 8 in IIIA Monograph Series. IIIA, 1999.
17. Eric Platon, Marco Mamei, Nicolas Sabouret, Shinichi Honiden, and H.VanDyke Parunak. Mechanisms for environments in multi-agent systems: Survey and opportunities. *Autonomous Agents and Multi-Agent Systems*, 14(1):31–47, 2007.
18. Juan A. Rodriguez-Aguilar, Pablo Noriega, Carles Sierra, and Julian Padget. Fm96.5 a java-based electronic auction house. In *Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology: PAAMS 97*, pages 207–224, 1997.
19. Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Networked group communication*, pages 30–43. Springer, 2001.
20. Paul Valckenaers, John Sauter, Carles Sierra, and JuanAntonio Rodriguez-Aguilar. Applications and environments for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):61–85, 2007.
21. Mirko Viroli, Tom Holvoet, Alessandro Ricci, Kurt Schelfthout, and Franco Zambonelli. Infrastructures for the environment of multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14:49–60, February 2007.
22. Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, February 2007.