Execution Infrastructure for Normative Virtual Environments

Tomas Trescak[a], Inmaculada Rodriguez[b], Maite Lopez Sanchez[b], Pablo Almajano[b]

[a]*Artificial Intelligence Research Institute, Spanish Council for Scientific Research, Barcelona, Spain*
[b]*Applied Mathematics Department, University of Barcelona, Barcelona, Spain*

Abstract

Virtual Institutions (VIs) have proven to be adequate to engineer applications where participants can be humans and software agents. VIs combine Electronic Institutions (EIs) and 3D Virtual Worlds (VW). In this context, Electronic Institutions are used to establish the regulations that structure interactions and support software agent participation while Virtual Worlds facilitate human participation. In this paper we propose Virtual Institution eXEcution Environment (VIXEE) as an innovative communication infrastructure for VIs. Using VIXEE to connect Virtual Worlds and EI opens EI to humans, providing a fully operational and comprehensive environment. The main features of the infrastructure are i) the causal connection between Virtual Worlds and Electronic Institutions, ii) the automatic generation and update of the VIs' 3D visualization and iii) the simultaneous participation of users from different virtual world platforms. We illustrate the execution of VIXEE system in a simple eAuction house example and use this example to evaluate the performance of our solution[1].

Keywords: Virtual Institutions, 3D virtual worlds, Virtual Environments
2010 MSC: 97R40

Email addresses: `ttrescak@iiia.csic.es` (Tomas Trescak), `inma@maia.ub.es` (Inmaculada Rodriguez), `maite@maia.ub.es` (Maite Lopez Sanchez), `palmajano@iiia.csic.es` (Pablo Almajano)

In memoriam, we acknowledge Dr. Marc Esteva for his valuable work in this research.

[1]We acknowledge ITMAS 2011 as the forum in which the main ideas behind this paper were preliminary discussed

1. Introduction and Motivation

Nowadays there is an increasing demand for e-* applications (where * stands for learning, commerce, government, etc.). These applications support the participation of humans that engage in different activities, to achieve their goals. Whenever some tasks can be delegated and automated, these applications can be enriched with software agents. As a consequence, human and agent interaction must be handled. The internet based and distributed software technologies, such as Virtual Worlds (VW) and Multiagent Systems (MAS), may support the engineering of this type of applications.

Specifically, we advocate to take a MAS approach for designing these systems and to use 3D Virtual Worlds [1] [2] to get humans-in-the-loop by facilitating their participation in the system. First, a 3D real-time representation of the system facilitates a better understanding of what is happening at both agent and the entire system levels. Second, thanks to the regulation imposed by the MAS, the 3D environment becomes a normative Virtual World, where norms are enforced at runtime. This automatic regulation contrasts with the way it is done in current virtual worlds where norms are restricted to the user's acceptance of the terms of service. Third, system participants can be both humans and software agents. In other words, it is an effective way to facilitate direct participation of humans in MAS, instead of just allowing them to customize agent templates with their preferences. This approach is taken in Virtual Institutions [3], which have proven [4] [5] [6] [7] to be an adequate platform to support this type of hybrid multi-agent systems, by combining Electronic Institutions (EI) [8], which is an Organization-Centered MAS (OCMAS), and 3D Virtual Worlds. In this context, Electronic Institutions are used to establish the regulations that structure interactions and support software agent participation while virtual worlds facilitate human participation. The former focuses on the definition of the institutional rules that structure participants interactions. The later is related to the 3D virtual world and supports immersive human participation on the system by controlling an avatar in a 3D representation of the institution.

The formal specification of an Electronic Institution establishes a common *ontology* and the *roles* participants may play. The *performative structure* defines the activities participants can engage on and the relationships among them. This is specified as a graph, where *nodes* are *scenes* and *transi-*

tions, and *arcs* connecting them are labelled with the roles that can progress through them. While scenes define interaction protocols (i.e. scene protocols) among participants by specifying the illocutions that can be uttered, transitions are used to model synchronization, parallelization, and choice points. Finally, *norms* define the consequences of agents' actions expressed as obligations.

A performative structure of an EI, is used by our Virtual World Grammar (VWG) [9] to automatically generate the virtual world design. Activities of performative structure are displayed as virtual spaces, while illocutions from the scene protocols are transformed to specific world interactions and gestures. Also, Virtual World Grammar mechanism allows to generate virtual world model for several different virtual worlds (e.g. Open Wonderland, Second Life).

Nevertheless, the generated design is only a 3D model of a virtual world, a visual layer, separated from the EI runtime infrastructure (called AMELI), the normative control layer. This separation does not allow normative control of world interactions at run-time. Thus, layers have to be connected, and in a way that would assure their consistent state according to the other layer. Therefore, the desired connection has to keep their causal dependence, when related actions in virtual worlds are processed by EI and related EI events are visualized in a virtual world, by manipulating the virtual world design (e.g. opening doors).

However, with the existence of such a large number of virtual worlds, it is often practical to let participate users from several different virtual worlds. This increases the possible user base or allows to perform experiments with different user groups (e.g. kids, teens or adults). In some cases, as in the case of the presented e-auction house, it is even desired to join the execution of the virtual world application with the non virtual environment, such as the web application or even the real world (see Section 5).

Combining several different environments and their simultaneous execution raises several inter-operability issues, such as:

1. Parallel presence, movement and interactions of avatars in different virtual worlds.
2. Virtual world participants that speak different languages try to participate in the same "single language" application.
3. Heterogeneous architectures of virtual worlds make it difficult to monitor and react to virtual world events and interactions, and to causally

update the virtual world model according to the EI state.

Considering point (1) solving parallel avatar presence and movement is out of scope of this research. However, we demand only a basic level of virtual worlds' interoperability. This means that participants from other virtual worlds are visualized as limited avatars, which only perform institutional actions (e.g. joining a scene). Thus, for instance, it is not shown how they walk around the room. An example application using multiple virtual environments is an auction where users from Second Life, Active Worlds or PS3 Network participate in the same auction room with a fixed amount of chairs. Hence, as soon as a participant takes one of the chairs, his avatar appears sitting in a chair, in all other universes (i.e. Virtual Worlds).

The popular solution to solve the problem of multilanguage environments, mentioned in point (2), is to define a common ontology, that each of the participating virtual worlds adopts for controlling and executing world's interactions. In our approach, we rely on Electronic Institutions, which define such common ontology for all institutional interactions.

Considering heterogeneous architectures in point (3), we need a mechanism that creates a mapping between virtual world dependent actions and institutional messages. In reverse, it has to define mappings between institutional events and target virtual environments, where such an event should be visualized. Concerning the content manipulation, the virtual world model is generated prior to the execution of the Virtual Institution, and then it is dynamically updated during the execution of the institution (e.g. launching of a scene in the normative layer can add a new room to an institutional building in the visual interaction layer). In our approach, we use Virtual World Grammar, which can dynamically manipulate 3D content of multiple virtual worlds.

Bogdanovych et al. presented the architecture of a causal connection server, which was able to create a causal connection between different environments (e.g. virtual world and mobile application) [10]. The drawback of this solution is a simple action-message table which makes it difficult to route events between different environments and an Electronic Institution. Therefore, we propose VIXEE as an innovative Virtual Institution eXEcution Environment which adds important extensions to previous Virtual Institution infrastructures. These extensions address generic and dynamic features. That is, VIXEE allocates at run-time participants from different VW worlds, and it modifies on the fly the content of a virtual world (e.g a new virtual room can be added during the execution of the infrastructure). An important factor of any middleware is its agility that is its ability to respond quickly and safely to both layers event during heavy loads. Therefore, we have evaluated our solution by measuring response time with a large number of agents (up to 500 agents).

VIXEE has already been successfully deployed in an e-learning scenario, the social historical simulation of the City of Uruk 3000 BC [7] [11] and an e-government scenario, a virtual market of water rights called vmWater [12] [13] [14]. We are working in another application deployed using VIXEE, it is a serious game for SmartGrid (electric grid) training [15].

The rest of the paper is organized as follows. Section 2 provides background information on concepts related to this research. Section 3 discusses related work. Then, in Section 4, we present VIXEE and explain in detail its implementation. In Section 5 we present a case study of an e-auction house application, using which we evaluate our system in Section 6. Finally, in Section 7, we give conclusions and state our future work.

2. Background

The concept of combining Electronic Institutions with 3D virtual worlds was introduced in [3] as Normative Virtual Worlds and named Virtual Institutions. In this context, Electronic Institutions are used to specify the rules that govern participants' behaviors, while 3D virtual worlds are used to facilitate human participation in the institution. Therefore, participants of Virtual Institutions can be both human and software agents. A Virtual Institution is separated into a *Normative Control Layer* and a *Visual Interaction Layer*. This provides support to the conceptual separation between the normative control of interactions and the design of the virtual world, i.e., the design of the 3D graphical user interface. The Normative Control Layer is responsible for the institutional control of interactions among participants, while the Visual Interaction Layer focuses on the 3D representation of the institution. Regarding participants, humans participate in the system by controlling an avatar on the Visual Interaction Layer. Software agents are directly connected to the Normative Control Layer, visualized as "special" avatars in the Visual Interaction Layer and participate by exchanging messages.

Both layers are causally connected, whenever one of them changes, the other one changes in order to maintain a consistent state [16]. In the case

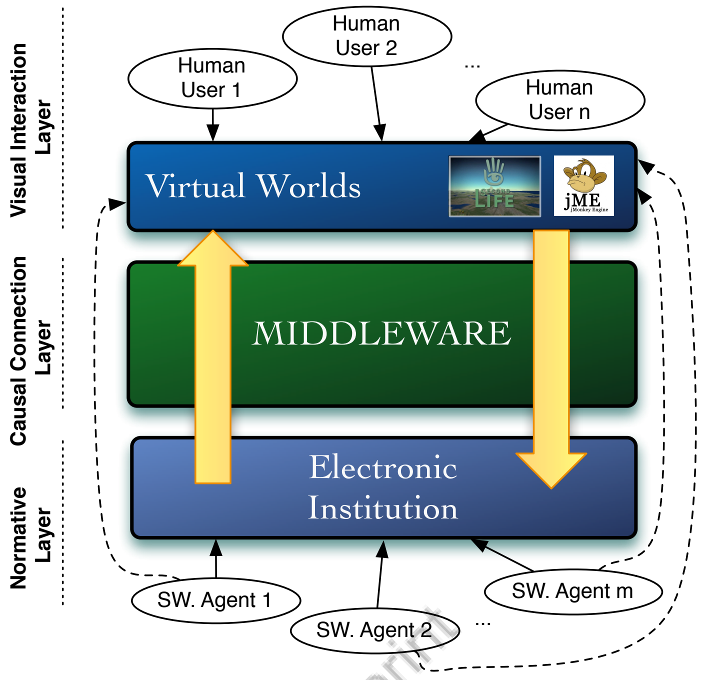


Figure 1: Overview of VI architecture.

of our Virtual Institution, a Causal Connection Layer keeps a consistent state between the model, represented by the Normative Control Layer, and its view, represented by the Visual Interaction Layer. Figure 1 shows an overview of the three layered architecture of Virtual Institutions.

There is an important conceptual difference between Electronic Institutions (EI) and Virtual Institutions (VI). In EIs everything is regulated in the sense that it is defined what is permitted, and everything else is prohibited. In VIs the situation is opposite: there are some actions provided by the virtual world platform that have institutional meaning, and hence, they are regulated (e.g. participants are obliged to pay for obtained goods before leaving the auction room), while the rest of actions are permitted (e.g. there is no regulation for walking around a room). This is similar to traditional organizations or institutions, where participants are able to perform many actions, but only some of them are regulated and have organizational or institutional meaning.

3. Related Work

In this research work, we advocate the use of Virtual Institutions as Normative Virtual Worlds for the deployment of e-* applications and social simulations. For the purposes of such applications, we found that current architectures of the middleware (Casual Connection Layer) lack features, therefore we proposed a new one. This section presents the state-of-the-art in causal connection architectures, as well as in works related to Normative Virtual Worlds.

First attempt to establish a *causal connection* between Electronic Institutions and 3D virtual worlds (in this case Adobe Atmosphere) led to the creation of Causal Connection Server (CCS) [5]. CCS implemented two basic causal actions: (i) creation of an external agent connected to AMELI when a new human user logs in the institution and (ii) visualization of a new avatar in a virtual world when a software agent connects to AMELI. Additionally, CCS used an action/message table mapping each virtual world action to an institutional message. Later, the same author presented an architecture that allowed causal connection between multiple environments (e.g. mobile devices, virtual worlds and the real world) and AMELI [10].

Subsequently, the implementation of the causal connection between a 3D virtual world and an Electronic Institution was further extended during the development of the Itchy Feet prototype [4] which used Virtual Institutions in a virtual tourism environment. For this purpose, authors implemented the Connection Server, bound to the Itchy-Feet implementation. Authors later removed the binding to their solution and presented a Generic Connection Server (GCS). The architecture of this system is depicted in Figure 2. GCS is a robust extension of CCS, that uses its own communication protocol between the virtual world and AMELI. Messages using this protocol are specified in XML format and validated against XSD schema. We have used the ideas from both GCS and CCS to design our new architecture.

Previous approaches to causal connection have some limitations: (i) they can only use a single virtual world (GCS) or the execution with multiple virtual worlds is difficult to control and maintain (CCS); (ii) they cannot manipulate the virtual world content at runtime; (iii) there is no mechanism for a fail-safe communication.

We have removed these limitations and created Virtual Institution eXEcution Environment (VIXEE). VIXEE is a generic solution, which handles the causal communication with multiple environments using the Movie

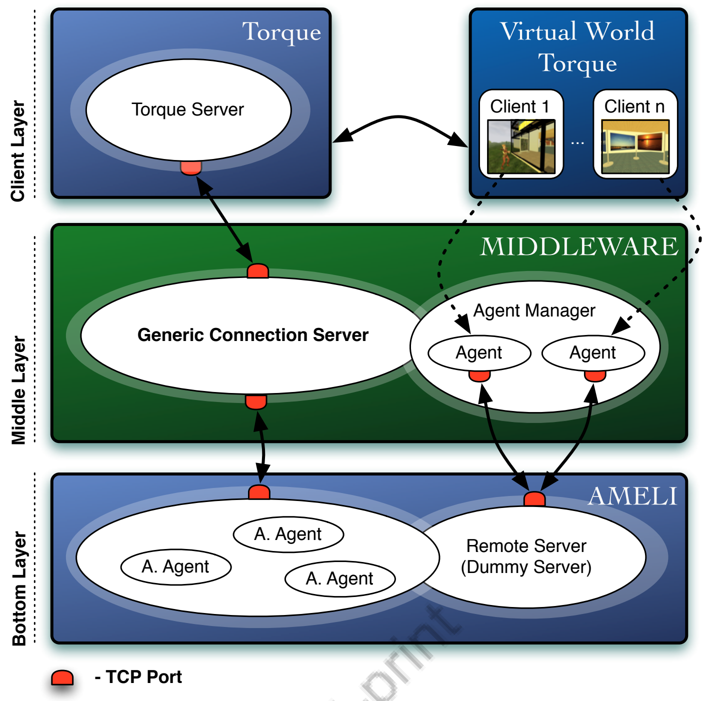


Figure 2: Architecture of the Itchy Feet solution

Script mechanism (see Section 4.2). Moreover, VIXEE dynamically reacts on changes in the institution and changes the virtual world content accordingly.

Cranefield et al. also aimed to structure social interactions in virtual worlds adapting techniques used for agents in MAS [17]. They monitored social expectations in Second Life virtual world. As a main difference, they used temporal logic rules and we use an Electronic Institution to define a set of rules that structure participants interactions.

Afterwards, in the same research line, authors addressed issues of collection, filtering and processing of sensor data [18]. This research led into the development of a tool that connects Second Life with Jason, a well known agent programming language and platform [19]. In this work, authors demonstrate a new framework in conjunction with an extension of the Jason BDI interpreter that allows agents to specify their expectations of future outcomes in the system and to respond to fulfillment and violations of these expectations. Meanwhile they take an agent-centered perspective connecting BDI agents to Second Life virtual world, we adopt an organizational approach proposing a generic infrastructure which casually connects a domain independent OCMAS (Organization Based MAS) with several virtual worlds.

The use of Normative Virtual Worlds has been explored in the context of computer games as well. In a previous research, we used Virtual Institutions in the definition of Quests for Massive Multi-player Online Games (MMOG) [20]. The infrastructure presented in this paper can be used to enforce the rules defined for a particular quest. In another work, Salceda et al. also modelled gaming scenarios using social structures [21]. In this context, agents (i.e Non-Player Characters) take into account high-level definitions that would allow for reasoning why to make a particular decision on a specific context. In comparison to this work, we use organizations to enforce agents' interaction rules at run-time, rather than extending individual agents' reasoning cycle with organizational awareness.

4. Virtual Institution Execution Infrastructure

In this section, we propose a Virtual Institution eXEcution Environment (VIXEE), which defines a new architecture of a middleware of a Virtual Institution, connecting users in the Visual Interaction Layer to the Normative Control Layer. Normative layer is in charge of regulating participant actions by means of AMELI, the EI execution infrastructure. VIXEE supports the participation of software agents by visualizing them as bots in connected Virtual Worlds; thus, human-software agent interaction is enhanced.

4.1. Solution Architecture

We design VIXEE respecting the 3-layered architecture of Virtual Institutions, depicted in Figure 3. The top layer consists of several 3D Virtual Worlds. The bottom layer is represented by the Electronic Institution execution environment (AMELI). Both layers are causally connected by our middleware, which consists of the Extended Connection Server (ECS) and the Virtual Worlds Manager (VWM).

4.1.1. Visual Interaction Layer

The Visual Interaction Layer consists of several 3D virtual worlds. Human users and selected software agents are represented by their avatars (i.e. 3D virtual characters). Each of the virtual worlds can be implemented in

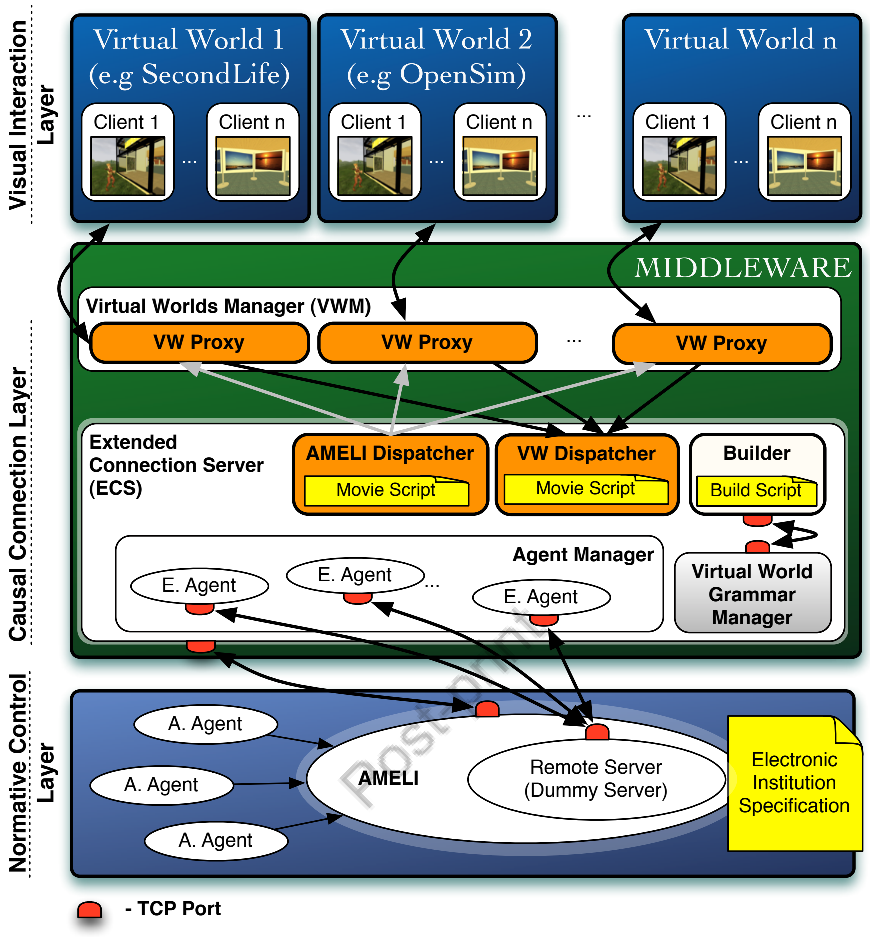


Figure 3: Architecture of the Virtual Institution Execution Environment

a different programming language and using different visualization technologies. The virtual world communicates with the middleware using a standard network protocol (e.g. TCP, HTTP).

The case study introduced in Section 5 uses OpenSim as the virtual world platform for the e-auction house Virtual Institution. The Visual Interaction Layer may be composed of different types of interfaces allowing users to participate from a 3D Virtual World as well as from a web page.

4.1.2. Normative Control Layer

The Normative Control Layer is represented by an Electronic Institution running in AMELI, which mediates agents' interactions while enforcing institutional rules. AMELI is a general-purpose MAS infrastructure, as it can interpret any institution specification generated by ISLANDER [22], the EIs specification editor. Therefore, it can be regarded as domain-independent. The combination of ISLANDER and AMELI provides full support for the design and development of Electronic Institutions [23]. For each participant within an institution, AMELI creates a governor, which mediates its participation in the institution, and coordinates with the rest of other agents in AMELI to guarantee the correct execution of the institution. Autonomous software agents (A. Agent in Figure 3) participating in the institution are directly connected to AMELI. AMELI uses two TCP ports to communicate with the middleware. The first one is used to announce changes in the institutional state (event) to the causal connection layer (e.g. started institution, agent entering or exiting a scene). The second one is used to receive messages, corresponding to VW actions, from the middleware.

4.1.3. Causal Connection Layer

The Causal Connection Layer (CCL) (hereafter we may refer to it as *middleware*) provides the causal connection between different 3D virtual worlds and an executing Electronic Institution. As Figure 3 shows, it is split into two main components: (i) Extended Connection Server (ECS), responsible for communication with AMELI; and (ii) Virtual Worlds Manager (VWM), responsible for communication with 3D virtual worlds. The rest of this section describes them.

Extended Connection Server (ECS)

ECS mediates all communications between AMELI and Virtual Worlds Manager. This layer introduces new features and improvements that were not available in previous designs of the Causal Connection Layer [5] [10] [4]. The most notable features are: support for multiple environments, reaction on early EI events, and connection fail-safe mechanisms. Current design allows us to connect one running instance of an Electronic Institution to multiple 3D virtual worlds, thus allowing joint participation of users from different virtual worlds in the same Virtual Institution. Reaction to early EI events allows to catch events like *institutionStarted*, which can trigger the construction of the virtual world design. The connection fail-safe mechanisms deal with connection errors from both virtual worlds and EI to the Casual Connection Layer.

An important part of ECS is the Agent Manager. For each human participating in a 3D virtual world, the Agent Manager creates an external agent (E. Agent in Figure 3) in the middleware representing him within the institution. Thus, when the avatar tries to perform an action that requires institutional verification, this agent is used to send the corresponding message to AMELI. Hence, AMELI perceives all participants as software agents. There are two different classes of external agents, one for human participants and another one for software agents. Currently, VIXEE contains our own model of external agents, programmed in .NET framework. External agents have the possibility to detect and react to events they receive from both virtual world and an Electronic Institution. Also, they connect to AMELI, where so called *governor* is created, which is responsible for monitoring and control of agents' actions.

Moreover, ECS includes the *Builder* component, which communicates with the Virtual World Grammar Manager, and it is responsible for manipulating the virtual world content. Specifically, the Builder uses the Virtual World Grammar to perform its tasks. Notice though, that whilst the Builder is responsible for creating a new geometry for a virtual world scene (e.g., virtual rooms), the Virtual World Manager is the component in charge of updating the virtual world visualization.

ECS uses the *VW Dispatcher* to mediate virtual world actions and the *AMELI Dispatcher* to mediate AMELI events. Both dispatchers use our proposed Movie Script mechanism to select which action to perform depending on the context of an action/event (see Section 4.2). As Section 4.3 details, when the VW Dispatcher receives an action from a virtual world, it sends the corresponding message to AMELI, and if necessary, waits for AMELI response (synchronous messages wait for a response, while asynchronous do not). The response to the event is communicated back to the virtual world (see Figure 5). 3D worlds are able to use either HTTP or TCP protocol to communicate its events. On the other hand, as described in Section 4.4, *AMELI Dispatcher* handles AMELI events. Such events provoke a causal change in the state of 3D virtual worlds, so AMELI Dispatcher triggers an action execution to each of the connected virtual worlds (see Figure 6).

ECS uses three TCP ports. First one is used to communicate between the Builder and Virtual World Grammar Manager. The second one is used to listen for AMELI events. The third one is used by the Agent Manager to send messages to AMELI (all external agents share the same port).

Virtual Worlds Manager (VWM)

Virtual Worlds Manager is used to mediate all communications between 3D virtual worlds and ECS and to dynamically manipulate the 3D representation of all connected virtual worlds. It consists of a set of virtual world Proxies, one for each connected virtual world (see Figure 3). These proxies allow to use a language specific for a given virtual world to communicate with VIXEE. Thus, for example, in Second Life we use OpenMetaverse[3] library to update the state of the virtual world. The content is manipulated after receiving a selected AMELI event (e.g. *SceneStarted*), when as a result a new room is created in the 3D virtual world. Next section details how this is done based on our *Movie Script* mechanism, which can map any event/action from a specific context (AMELI event or virtual world action) to a *Movie Script* action.

4.2. Message Handling: Movie Script Mechanism

Virtual World interactivity is accomplished through *virtual world actions* that human users and software agents perform within a virtual world. Such actions are either *institutional* or *non institutional*. An institutional action has to be validated by the Electronic Institution running in AMELI (e.g. entering some scene). On the contrary, execution of non institutional actions does not have to be validated by the institution (e.g. walking).

When an institutional action is performed in a virtual world, an Electronic Institution needs to be causally updated by receiving the corresponding AMELI message (e.g. agentEnterScene). In reverse, when the Electronic Institution produces an event (we refer to such an event as *AMELI event*), all virtual worlds need to be causally updated. The original mechanism that handles this mechanism, proposed by [5], is an Action/Message table. This table holds mappings between virtual world actions and an institutional message. An example from the e-auction house example is the mapping of the gesture raising hand to the institutional message "bid". The problem arises when we need to assign two different meanings for the same gesture, or the same action. That is, actions or gestures can have different meaning in a different context (e.g. police man raises hand to stop traffic). Another is-

[3]http://openmetaverse.org/, (last accessed 05/2012)

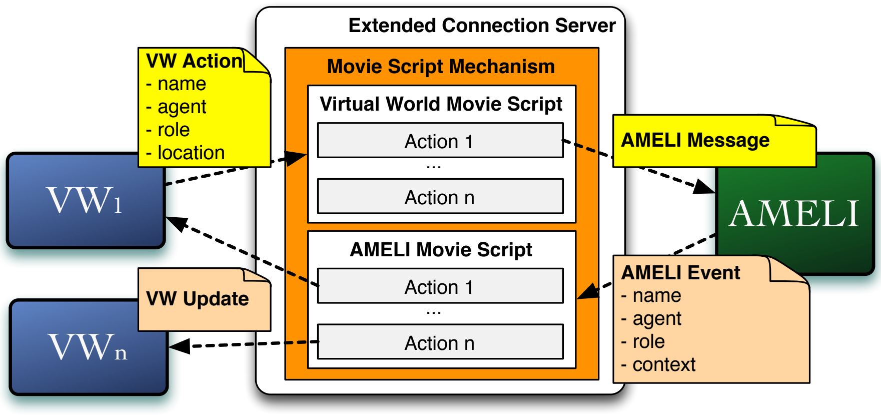


Figure 4: Movie Script Mechanism Conceptualization

sue of the original approach is that it is not possible to route events to and from multiple environments, meaning, that we cannot decide which events to process in which environment and in what manner.

Thus, we propose our *Movie Script* mechanism to process the transformation of virtual world actions to AMELI messages and in reverse transformation of AMELI events to virtual world updates. Figure 4 describes how the Movie Script mechanism uses *Movie Script actions* to handle such transformations. This mechanism supports the virtual world independence and facilitates a simple and consistent definition of the expected behavior of a 3D virtual world. A virtual world Movie Script is defined for all virtual environments and a AMELI Movie Script for an Electronic Institution. Both are processed by the VIXEE environment. A Movie Script contains a number of lines. Each virtual world Movie Script line specifically defines the context for which this Movie Script line applies (e.g. participant role, virtual world location, action name). Each AMELI Movie Script line defines for which AMELI event this line applies (e.g. event name, institution name, sender role) and which virtual worlds should be notified about this event using what Movie Script action.

Virtual World Actions

VIXEE differentiates between three types of institutional actions: *illocutionary* (illocutions that agents try to utter within scenes), *motion* (move-

Action Type	Action	Description
Motion	enterInstitution	Request to enter the institution
	moveToTransition	Request to move from a scene to a transition
	moveToScenes	Request to move from a transition to several scenes
Illocutionary	saySceneMessage	Request to say a message in a scene
Information Request	accesScenes	Ask for the scenes the agent can join from a transition
	accesTransitions	Ask for the transitions the agent can join from a scene
	agentObligations	Ask for pending agent's obligations
	sceneState	Ask for a scene's current state
	scenePlayers	Ask for agents in a scene

Table 1: Institutional Actions

ments between scenes and transitions), and *information request* (scenes reachable from a transition, transitions reachable from a scene, agent's obligations, scenes' states, and scenes' participants).

Institutional actions are initiated in a specific virtual world and validated by sending the corresponding message to its Proxy and observing the response. Table 1 shows examples of institutional actions. The *Movie-script mechanism* maps the institutional action to the defined *Movie Script action*, which depending on the action context received from the Proxy, creates the AMELI message and sends it to AMELI. For each received message, AMELI replies with one of the following messages: *agree* (acknowledge message), *refuse* (incorrect message), or *unknown* (message not understood).

We seek inspiration in a movie production, where depending on a current scene and its state, an actor performs an action. Like a regular Movie Script, it contains script lines. Each line holds the definition of the context, upon which a defined action is executed. Formally:

Definition 1. Virtual World Movie Script is a function:
$vwf : A \times R \times L \times VWA \rightarrow MSA$ which maps a virtual world action $vwa \in VWA$ to a corresponding Movie Script action $msa \in MSA$ depending on its context, where:

1. A is a set of Virtual Institution participants

2. R is a set of roles that participants can take while participating in the Virtual Institution
3. L is a set of locations, that is either transitions or scenes
4. VWA is a set of virtual world actions being sent from the virtual world
5. MSA is a set of Movie Script actions

An example of a script line from the eAuctions institution (see Section 5) is: vwf_1(agentId, buyer, auctionRoom, saySceneMessage("Bid", AgentId, amount)) = makeBid(AgentId, buyer, auctionRoom, saySceneMessage("Bid", AgentId, amount)). This lets a user with role buyer bid a specific amount of money and sends the bid request to AMELI.

AMELI events

The causal connection of AMELI with all connected virtual worlds is maintained by the AMELI Dispatcher, which reacts to received AMELI events by executing a corresponding Movie Script action. Then, for each of the connected virtual worlds, and for each of the relevant AMELI events, we have to implement a Movie Script action, executed by the corresponding Proxy, which updates the virtual world visualization. Table 2 contains examples of typical AMELI events. Formally:

Definition 2. AMELI Movie Script is a function:
$amf : E \rightarrow \{VW \times MSA\}^*$ which depending on received AMELI event $e \in E$, for each of the connected virtual worlds $vw \in VW$ returns a corresponding Movie Script action $msa \in MSA$ where:

1. E is a set of AMELI events, where each event contains the name of the event, the event context (identifying the origin of the event) and message specific content.
2. VW is a set of identifiers of all connected virtual worlds.
3. MSA is a set of Movie Script actions. Specifically $msa \in MSA$ is a Movie Script action which updates the given virtual world $vw \in VW$ depending on the content of the message $e \in E$. In particular, msa can be empty, if no Movie Script action is defined for a specific virtual world.

An example of an AMELI Movie Script line from the e-auction house institution (see Section 5) for AMELI event $ae_1 = (agentEnteredScene[agent = buyer_1, scene = Auction])$ is: $amf_1(ae_1) = \{\text{SecondLife, openDoor}(ae_1)\}, \{\text{OpenSim, openDoor}(ae_1)\}$. This opens the door of the *auction* scene in Second Life as well as Open Simulator.

4.3. VW Actions Implementation

There are different activities that participants perform in 3D virtual worlds. For example, they interact with objects and communicate with other participants. It is similar in Virtual Institutions where agents represented by avatars must be able to perform institutional actions. In this section, we present how our system handles 3D virtual world actions, maintaining the causal dependence with an Electronic Institution using a bidirectional connection.

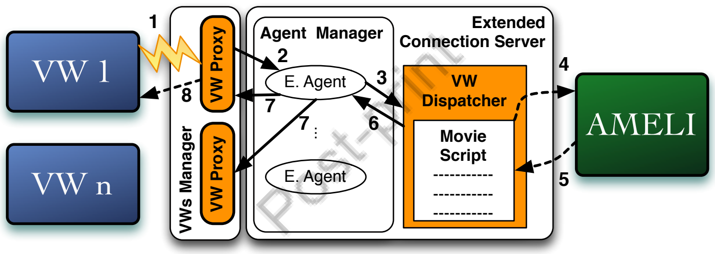


Figure 5: Message flow for VW generated action. Dashed lines represent the *message* passing, solid lines represent the *use* relationship.

Figure 5 depicts the message flow of a virtual world action. In this Figure, the dashed line represents a *message*, while solid line represents the *use*[4] relationship. When an action happens in a virtual world, it is sent to the Connection Server using a corresponding VW proxy (1). Next, the received message is passed to the external agent representing the user who has performed the action (2). The external agent uses the VW Dispatcher to send the message to AMELI. Specifically, VW Dispatcher uses the Movie Script mechanism (function vwf) to find the action (3). The Movie Script action is executed, and VW Dispatcher sends the corresponding message to AMELI

[4]We consider the UML 2.0 notation

(4). Then, agent actions are evaluated by AMELI. As a result, an AMELI event may be generated (see Section 4.4), and all connected virtual worlds are notified about agents' actions. For an asynchronous message, the process ends here.

For synchronous messages, the ECS waits for the response, which is either the confirmation of the action execution or an error message (5). The response is sent to the Agent Manager (6). Finally, The Agent Manager contacts all the connected VW proxies (7) and each one of them informs its related 3D virtual world about the result of the action (8). If the result is positive, the action is visualized in the 3D virtual world. Actions produce human readable or visual output so the participant can perceive the output of his action either by a change in a 3D world (opening door) or by receiving a message.

4.4. AMELI Events Implementation

AMELI keeps the execution state of an EI, which evolves as consequence of participant actions. In such a case AMELI informs participants about those changes by using a set of institutional events (i.e. AMELI events). Examples of such events are: institution started, scene started, agent entering or leaving a room or a transition. VIXEE provides support for all AMELI events.

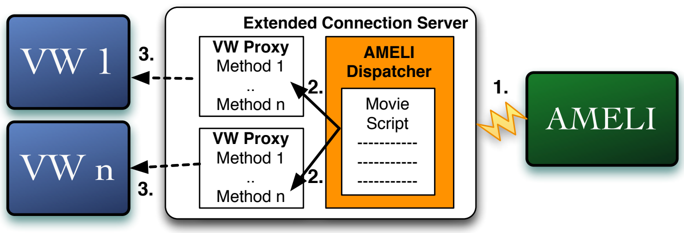


Figure 6: Message flow from AMELI to 3D virtual world

Figure 6 depicts how VIXEE processes an AMELI event. When AMELI triggers an event (1) it is communicated to ECS using TCP in the predefined AMELI format. AMELI has a fixed, predefined set of events. When ECS receives an event, it calls the AMELI Dispatcher that uses the Movie Script function amf to look for an action msa to execute (2). Then, for each of the connected VW Proxies, it dispatches the related action (3). If requested, ECS

Event	**Description**
InstitutionStarted	An instance of new Electronic Institution was created
SceneStarted	An instance of new scene was created in AMELI
SceneFinished	An instance scene was destroyed
EnteredAgentEI	An agent has entered an institution
ExitedAgentEI	An agent has exited the institution
MovedToScene	An agent has moved to a scene
ExitedScene	An agent has been exited of the scene
SaidMessage	A message has been said in a scene

Table 2: Examples of AMELI events

can use the *Builder* to dynamically create and update the 3D representation of the related virtual world.

Table 2 contains the list of typical AMELI events. Notice that some events require the use of Builder component to manipulate the virtual world content. These events and their corresponding VW updates are:

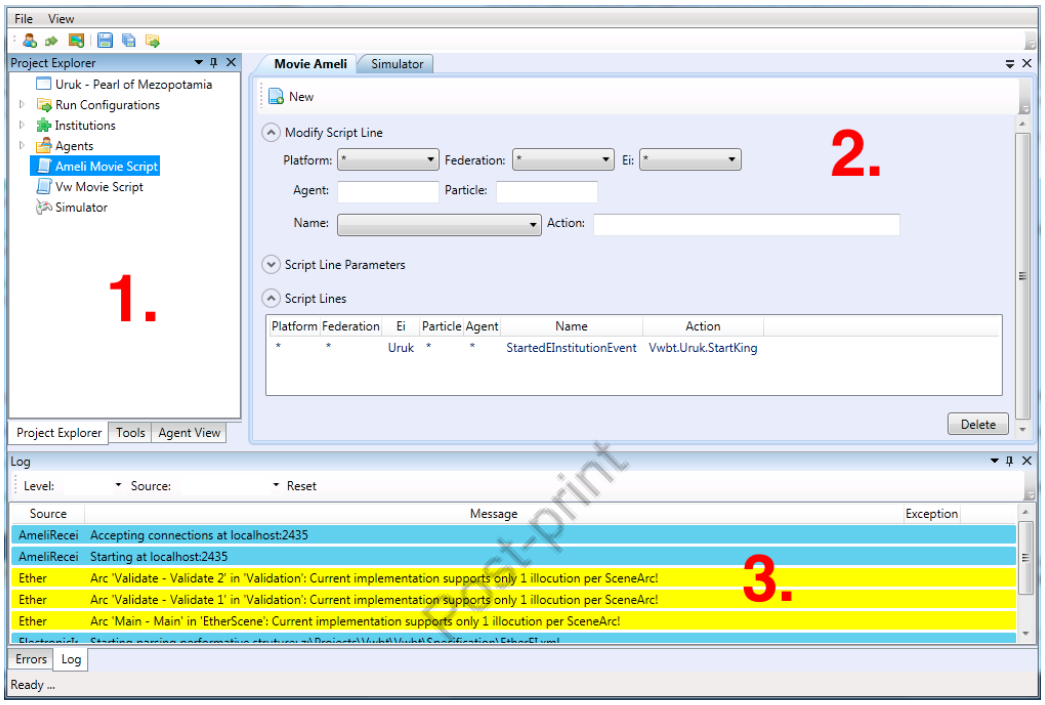
- *InstitutionStarted* - The system generates a 3D representation of an institution from scratch or resets the institution to the default state.

- *SceneStarted* - The system generates a 3D representation of the scene and sets the virtual world parameters so that it is possible to enter the generated scene.

- *SceneFinished* - The system removes a 3D representation of the scene and teleports all avatars out of the scene.

4.5. VIXEE Interface

While previous systems (CCS [5], GCS [4]) needed to run several components and programs in order to connect a Virtual Institution to a virtual world, VIXEE runs as a stand-alone tool with its own user graphical user interface. Figure 7 shows VIXEE interface demonstrating its parts. The project explorer (1) contains all parts of VIXEE infrastructure, including Electronic Institutions, agents and Movie Scripts. Simulator is used to launch agents within the virtual world. Project item editor (2) allows to modify selected project part using a custom visual interface. Figure 7 shows the AMELI movie script editor. (3) Log component describes application log messages. It is also used to monitor the incoming and outgoing communication.



Modify Script Line

Platform: * Federation: * Ei: *

Agent: Particle:

Name: Action:

Script Line Parameters

Script Lines

Platform	Federation	Ei	Particle	Agent	Name	Action
*	*	Uruk	*	*	StartedEInstitutionEvent	Vwbt.Uruk.StartKing

```
Source       Message                                                                                         Exception
AmeliRecei   Accepting connections at localhost:2435
AmeliRecei   Starting at localhost:2435
Ether        Arc 'Validate - Validate 2' in 'Validation': Current implementation supports only 1 illocution per SceneArc!
Ether        Arc 'Validate - Validate 1' in 'Validation': Current implementation supports only 1 illocution per SceneArc!
Ether        Arc 'Main - Main' in 'EtherScene': Current implementation supports only 1 illocution per SceneArc!
ElectronicIr Starting parsing performative structure: z:\Projects\Vwbt\Vwbt\Specification\EtherEI.xml
```

Figure 7: VIXEE interface

5. Case Study: eAuction House

Our case study concerns an auction system which allows both in-house users (bidders present in a real auction room) and internet users to participate in real auctions happening all around the world. This proved to be useful for specific types of auctions, like fish market auctions [24], happening over a short time period, in the exact hour on the exact place. These types of auctions use extensive visual information for auctioneers, e.g., fish quality, that decide the final price of the auctioned item. However, how to accomplish the presence in all these places and achieve an effective and comfortable communication between in-house and internet users? Our answer is a hybrid environment which combines 3D virtual worlds, augmented reality and multi-agent system technologies. We generate an auction as a virtual space, either as a room in a big auction building or a separate building in the virtual world. All auction participants are displayed as avatars. Internet users move around the building and visit different auctions by entering auction rooms. In-house users are tracked either by cameras or some communication device, and their actions are constantly updated in the 3D representation. The auction system displays auction progress to every type of participant in the following format:

- Each in-house user sees an announcement board with the auction progress, or this information is displayed in custom glasses that he wears, allowing to augment reality.
- Web user sees a dynamic page control displaying the auction progress.
- Virtual world user observes the actual environment and the behavior of avatars to see the auction progress.

First, to define the Virtual Institution supporting the e-auction house execution, we specify the normative control layer of the Virtual Institution, which is an Electronic Institution specification. Figure 8 depicts a performative structure (set of connected scenes) of the e-auction house institution. As this figure depicts, institution supports roles of seller, buyer, staff and auctioneer. Staff agent is a software agent responsible for automatic processes, such as the creation of an auction room and controlling the auction execution protocol. To start the auction, staff agent changes role to auctioneer. Scenes in this institution are: *ItemInfo*, *ItemRegister* and *Auction*. *Initial* and *Final* scenes are specific scenes through which participants can enter or leave the institution.

In this specification, sellers first register items in an *ItemRegister* scene. If no auction is running, staff agent creates a new auction scene and waits for participants to start the auction. Buyers join the *ItemInfo* scene to check the list of items to be auctioned and the auction starting time. When a required number of buyers enters the auction scene, the auction starts. Buyers can bid either by raising their hand, or by typing the *bid* command. When the auction finishes a winner cannot leave the auction room until he pays for the items he won. When no more items are to be auctioned, the auction room is destroyed (removed from virtual space).

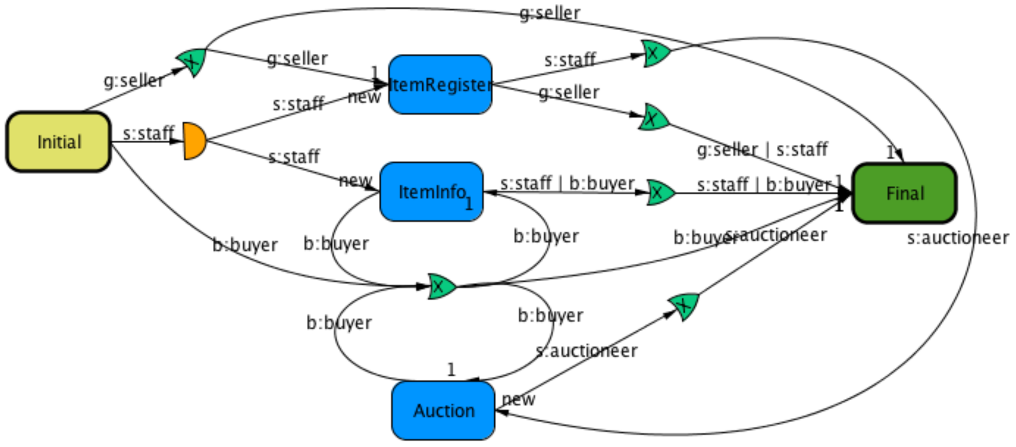


Figure 8: Performative structure of the eAuctionHouse institution

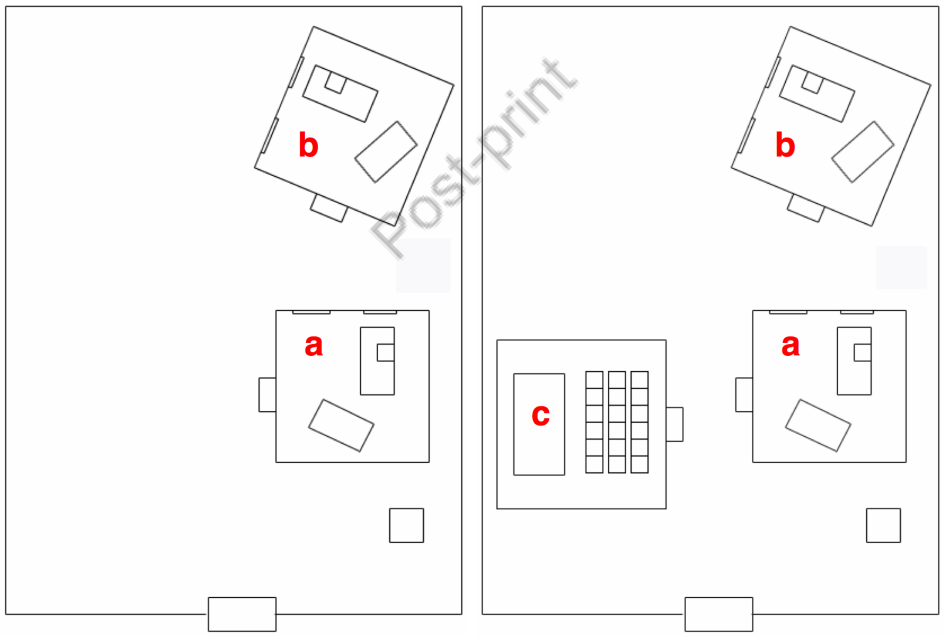


Figure 9: Initially generated floor plan (left) and the floor plan generated with the addition of the auction room (right)

Second, using the Virtual World Builder Toolkit, we define a Virtual World Grammar that allows us to generate the 3D representation of the

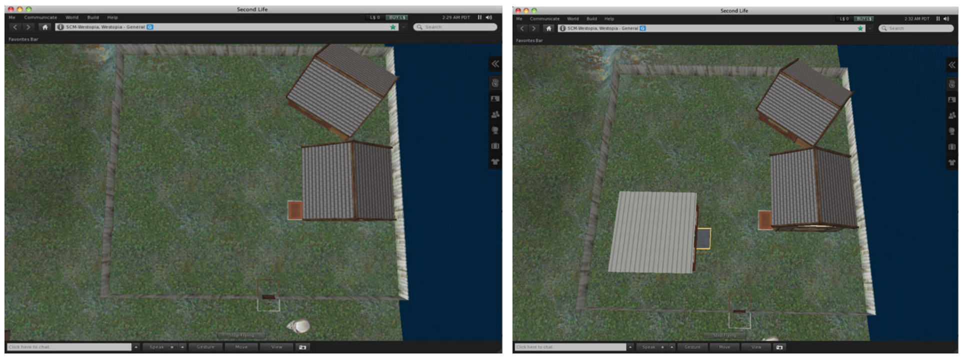
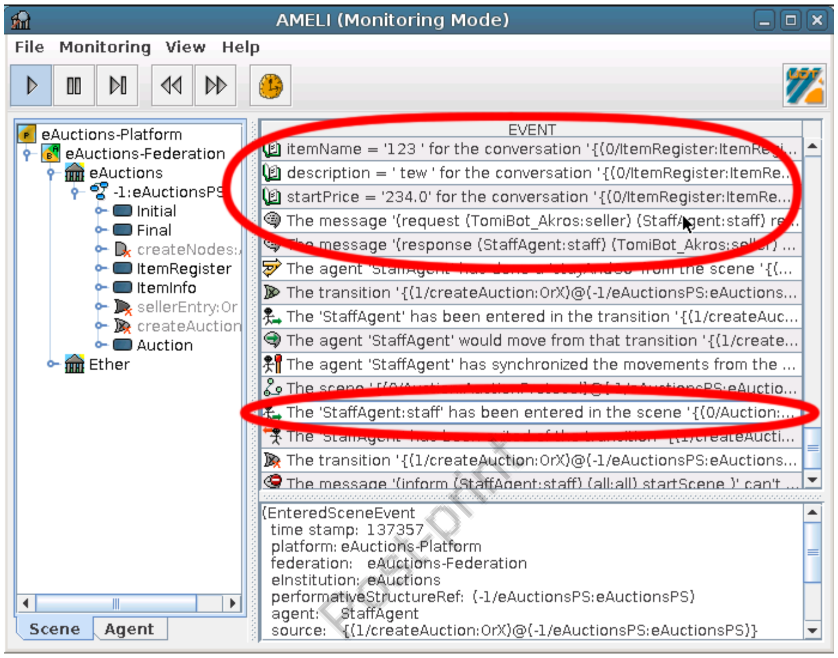


Figure 10: Initially generated 3D model (left) and the 3D model generated with the addition of the auction room (right)

Virtual Institution from the performative structure. In this tool, we can test the generated output. An example of the generated floor plan is displayed in Figure 9. To generate this floor plan, we map *ItemRegister* scene to the registration room, *ItemInfo* scene to the item information room and *Auction* scene to the auction room. In this Figure 'a' is the registration room, 'b' is the item information room, and 'c', on the right side of the figure, represents the subsequently generated auction room.

Finally, once the Electronic Institution and Virtual World Grammar are specified, we map virtual world actions and AMELI events to the corresponding *Movie Script actions*. An example of virtual world action is an avatar raising his hand in the auction room that triggers a bidding Movie Script action. The bidding action is an example of the synchronous event, where its success or failure is visually announced to the other participants in the virtual world. We also map an AMELI event of a StaffAgent creating a new Auction scene to the Movie Script action where the system adds a new auction room to the visualization of the virtual world, where avatars will participate in the auction.

Once previous steps are defined, we can run VIXEE. Figure 9 depicts floor plans generated by the Virtual World Grammar, while the left part of Figure 10 depicts a visualization of the eAuction House in Second Life after launching the institution and with *ItemRegister* and *ItemInfo* scenes. The right part of Figure 10 depicts the aerial view of the institution after the auction room has been generated. Figure 11 shows the AMELI interface: a top set of highlighted lines indicate that a seller agent registered a new item for the auction; a single highlighted line shows an event describing that staff agent entered (and thus created) a new Auction scene. Additionally, Figure 12 shows avatars participating in the running auction, from the point of view of the auction manager.



```
itemName = '123 ' for the conversation '{(0/ItemRegister:ItemRegi...
description = ' tew ' for the conversation '{(0/ItemRegister:ItemRe...
startPrice = '234.0' for the conversation '{(0/ItemRegister:ItemRe...
The message '(request (TomiBot_Akros:seller) (StaffAgent:staff) re...
The message '(response (StaffAgent:staff) (TomiBot_Akros:seller) ...
The agent 'StaffAgent' has done a StayAndGo from the scene '{(...
The transition '{(1/createAuction:OrX)@(-1/eAuctionsPS:eAuctions...
The 'StaffAgent' has been entered in the transition '{(1/createAuc...
The agent 'StaffAgent' would move from that transition '{(1/create...
The agent 'StaffAgent' has synchronized the movements from the ...
The scene '[(0/Auction:AuctionProtocol)]@[-1/eAuctionsPS:eAuctio...
The 'StaffAgent:staff' has been entered in the scene '{(0/Auction:...
The 'StaffAgent' has been exited of the transition '{(1/createAucti...
The transition '{(1/createAuction:OrX)@(-1/eAuctionsPS:eAuctions...
The message '(inform (StaffAgent:staff) (all:all) startScene )' can't ...
```

```
(EnteredSceneEvent
 time stamp:  137357
 platform: eAuctions-Platform
 federation:   eAuctions-Federation
 eInstitution: eAuctions
 performativeStructureRef:  (-1/eAuctionsPS:eAuctionsPS)
 agent:    StaffAgent
 source:    {(1/createAuction:OrX)@(-1/eAuctionsPS:eAuctionsPS)}
```

Figure 11: AMELI interface

6. Evaluation

In this section, we evaluate the performance of VIXEE using the e-auction house Virtual Institution example. Participants of this institution are humans and software agents. Humans participate playing the roles of buyers and sellers. Software agents play either as staff, or as buyers or sellers. During the test we simulate virtual world actions of humans and software agents and measure VIXEE's response time to such actions (e.g. VIXEE's response time to validating the entrance to a particular scene or response time to a bidding action). The simulation of actions is done by feeding VIXEE's proxy with VW event calls (step 1 in Figure 13) and measuring time of receiving a response (step 8 in Figure 13). This test was repeated ten times recording obtained values. We have measured this response time in three intervals:

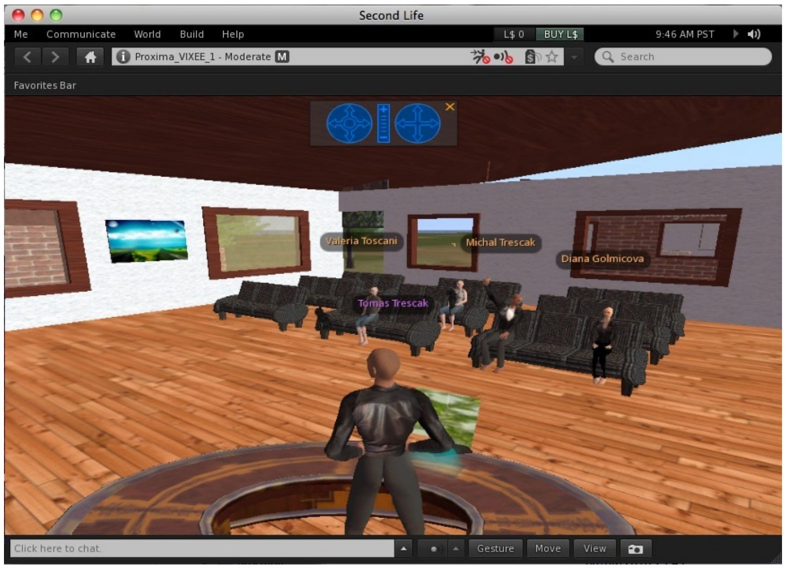


Figure 12: Avatars participating in an ongoing auction

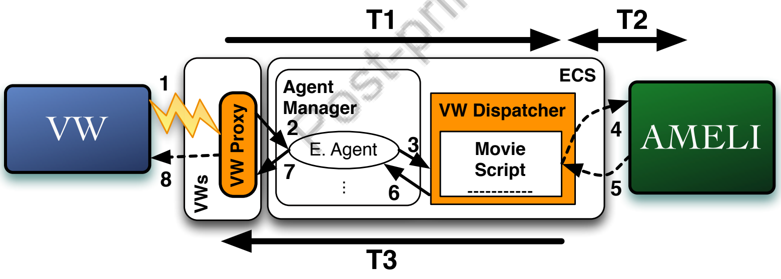


Figure 13: Measured response time intervals

- **T1** is the time interval between receiving the message from virtual world and sending it to AMELI server (that is time of step 1, step 2 and step 3 in Figure 13). In this interval VIXEE parses the message from text format into the form understood by VIXEE, finds appropriate Movie Script action and executes it and depending on the result of the action it then sends this message to AMELI.
- **T2** is the time interval between sending and receiving the message from AMELI (that is time of step 4 and step 5 in Figure 13).

- **T3** is the time interval VIXEE needs to process the received AMELI response and to send it back to the virtual world (that is time of steps 6, 7 and 8 in Figure 13).

To simulate actions of humans and software agents, we have created two different sets of actions (i.e. plans) that VI users typically perform within the eAuction institution. First plan is performed by the simulated human user with the *buyer* role. In this plan, the human user enters the institution, obtains the list of items registered for an auction, and then enters the auction. In order to get the list of auctioned items, the human avatar communicates with the *staff* agent. Second plan is executed by a simulated software agent with the role of *seller*. We assume that a human user programmed the software agent to register the items for him. In this plan, agent enters the institution, registers the item and then leaves the institution.

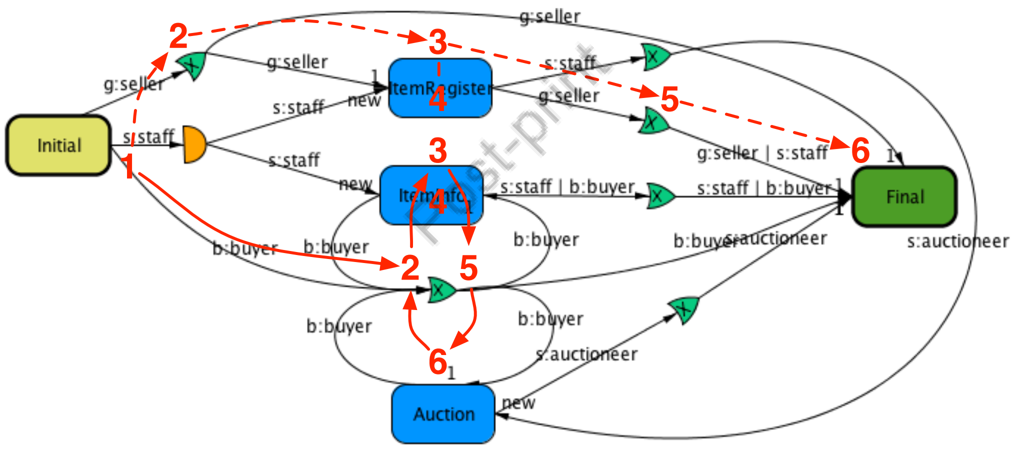


Figure 14: Two different plans that VI participants follow during the test (dashed lines are for SW agents and solid lines for human users).

Figure 14 shows the actions that follow both simulated humans and software agents. In the plan for simulated human users (marked with solid lines), humans enter the institution and move to the *Initial* scene (1). Then, they exit the *Initial* scene (2) and move to the *ItemInfo* scene (3) where they request the information about currently auctioned items (4). Then, they exit the *ItemInfo* scene (5) and move to the *Auction* scene (6). After moving to the *Auction* scene, agents decide not to participate, so they exit the scene

(2) and move back to *ItemInfo* scene (3). This creates an infinite loop of actions. In step (4) we simulate the execution of a complex Movie Script action (e.g. finding specific auctioned items, considering a huge amount of registered items). We set the execution time of this action to 1500 ms. We use this simulated action to prove that scalability of VIXEE depends only on the implementation of the specific VI that is, its Movie Script actions.

The plan for software agents in Figure 14 is marked with dashed lines. In this plan agents enter the institution and move to the *Initial* scene (1). Then, they exit the *Initial* scene (2) and move to the *ItemRegister* scene (3), where they register some items (4). Then, agents leave this scene (5) and exit the institution (6). When their plan is completed, agents restart it.

We ran the test in multiple threads where each thread ran a predefined number of agents. The test randomly decides how many human users and how many software agents will simulate. Threads run in parallel, where each thread executes actions in the following manner: (i) randomly select an agent; (ii) execute one step from agent's plan; (iii) wait; then thread waits a random time from an interval of $[0, 3]$ seconds. Executions of random agents in random time simulates real-world behavior where different actions from different agents are executed simultaneously.

To evaluate the system scalability we ran two different tests with different amounts of threads and agents and compared the results. We tested the system response time from two different aspects:

- Testing the average response time separated the presented three intervals (T1-T3) of each of the steps of the plan (1-6), for 100 and 500 agents.
- Testing the total average response time while incrementally increasing the number of active agents

	T1	**T2**	**T3**	**Total**
With step 4	87,79 ms	2,92 ms	0,03 ms	90,75 ms
Without step 4	2,22 ms	3,18 ms	0,04 ms	5,45 ms

Table 3: Average response times for 100 agents (in milliseconds)

First test ran with 10 threads, each running 10 agents (100 agents in total). First row of the Table 3 shows the average response time of each

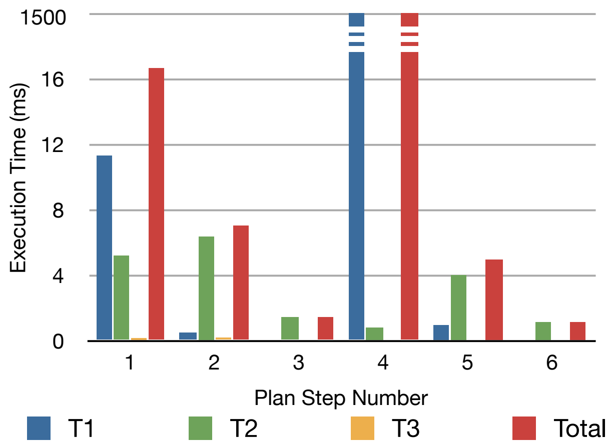


Figure 15: Average step execution time for all steps for 100 agents

interval T1, T2 and T3, along with an average *total* response time. Second row of the Table 3 shows the average response times without the step 4 (i.e. complex Movie Script action). Then, we see that the average response time drops to 5 ms. This shows that the limits of VIXEE are bound to the complexity of the Movie Script actions, which is domain dependent, since it corresponds to the implementation of the specific Virtual Institution. To further illustrate this, Figure 15 shows the average execution times for each of the six actions for 100 agents, clearly showing that action 4 takes the longest execution time.

	T1	**T2**	**T3**	**Total**
With step 4	91,54 ms	6,83 ms	0,05 ms	98,44 ms
Without step 4	8,81 ms	6,86 ms	0,05 ms	15,73 ms

Table 4: Average response times for 500 agents (in milliseconds)

In the second test, we ran 25 parallel threads, each running 20 different agents, that is 500 agents in the same virtual environment. Agents were joining VIXEE in zero to three seconds interval. First row of the Table 4 shows the average time of each interval T1, T2 and T3, along with the average *total* response time with all six actions included. Second row of the Table 4 shows the average response time without step 4 (that is the step where we perform a Movie Script action taking 1500 ms). By comparing both tables, we can see, that even that we have increased the number of agent five times,

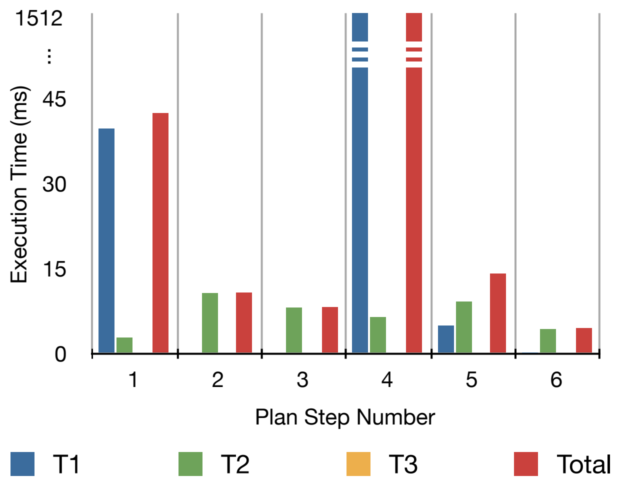


Figure 16: Average step execution time for all steps for 500 agents

VIXEE's average response time increased from 5 ms to 15 ms (making the relation sublinear). Figure 16 shows the average execution times for each of the steps. We can note that in comparison with the times from Figure 15, the sublinear relation is kept for all actions.

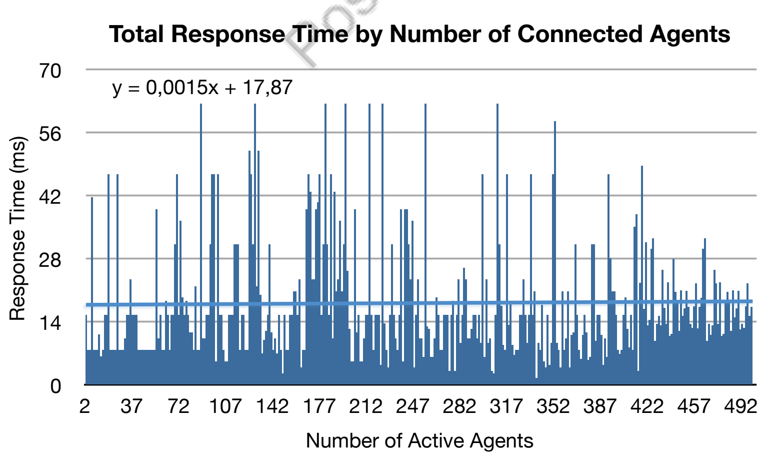


Figure 17: Average response time with different number of connected agents

Another aspect that we have evaluated was the total response time ($T1 + T2 + T3$) during the incremental load. We have been adding agents one by one till we were running 500 agents. We let all agents execute some actions in parallel, and we measured the VIXEE's response time. Figure 17 shows the graph of average response times depending on the number of connected agents. We can observe that even with the very high number of connected agents VIXEE was slowing down steadily with the average response time around 20 ms. We have approximated the performance decrease by a linear function $y = 0,0095x + 15,34$, that just corresponds to 0.9% performance decrease by each connected agent. The computed coefficient of determination is $R^2 = 0.0003$[5].

7. Conclusions

In this paper we have presented VIXEE, the Virtual Institution Execution Environment, which is an innovative communication infrastructure for Virtual Institutions where participants are human and software agents. Virtual Institutions are normative 3D virtual worlds where participants interact to achieve their common or individual goals. The main contributions of our research are:

- Our design of a middleware layer, which provides causal connection of several virtual worlds with an Electronic Institution which uses our Movie Script mechanism, improving previous version of an Action/Message approach. This allows users from different virtual worlds to participate in the same institution.

- Combination of our middleware with Virtual World Grammar allows dynamic manipulation of an environment content in different environments.

This paper describes the architecture and communication processes of VIXEE and explains what changes were made in comparison to previous approaches [5] [10] [4]. VIXEE provides an *architecturally neutral*, *domain independent*, and *scalable* solution for causal connection in Virtual Institutions. Architectural-neutrality from the agent point of view is given by AMELI, allowing to execute heterogeneous agents with any architecture. From the Virtual World point of view, a Virtual World Manager allows to connect different Virtual Worlds with any architecture. Domain independence is supported by

[5]The y function and R^2 function were computed by the standard functionality of the Numbers program for Mac

Virtual Institutions concept and the Movie Script mechanism, where VIXEE uses movie script to handle communication in Virtual Institutions from many domains. We have presented the e-auction house institution example to contemplate the dynamic update of the 3D model of this institution.

We evaluated the performance of VIXEE in a high load scenario, with 500 agents executed in 25 threads. We have measured average VIXEE response time, while increasing the number of connected and communicating agents. We conclude that VIXEE does not introduce any limitations on the scalability of the system, and it is only limited by the complexity of the implementation of the movie script actions and by scalability limits of selected virtual worlds.

As future work we plan to evaluate our VIXEE in different scenarios from e-* (e-commerce, e-government) applications. We also plan to evaluate the usability of the system with human users using different virtual worlds.

References

[1] R. Bartle, Designing Virtual Worlds, New Riders Games, 2003.

[2] P. R. Messinger, E. Stroulia, K. Lyons, M. Bone, R. H. Niu, K. Smirnov, S. Perelgut, Virtual worlds - past, present, and future: New directions in social computing, Decision Support Systems 47 (3) (2009) 204–228.

[3] A. Bogdanovych, H. Berger, C. Sierra, S. Simoff, Humans and agents in 3D electronic institutions, in: F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, M. Wooldridge (Eds.), 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands, ACM, ACM, 2005.

[4] I. Seidel, Engineering 3D virtual world applications design, realization and evaluation of a 3D e-tourism environment, Ph.D. thesis, Technischen Universitat Wien Fakultat fur Informatik (2010).

[5] A. Bogdanovych, Virtual institutions, Ph.D. thesis, University of Technology, Sydney, Australia (2007).

[6] A. Bogdanovych, S. Simoff, M. Esteva, Virtual institutions prototype, in: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, AAMAS '09, International

Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2009, pp. 1373–1374.

[7] A. Bogdanovych, J. A. Rodríguez, S. Simoff, A. Cohen, C. Sierra, Developing virtual heritage applications as normative multiagent systems, in: Proceedings of the 10th international conference on Agent-oriented software engineering, AOSE'10, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 140–154.

[8] M. Esteva, Electronic institutions: From specification to development, Ph.D. thesis, Artificial Intelligence Research Institute (IIIA-CSIC), Spain (2003).

[9] T. Trescak, M. Esteva, I. Rodriguez, A virtual world grammar for automatic generation of virtual worlds, The Visual Computer 26 (2010) 521–531.

[10] A. Bogdanovych, S. Simoff, M. Esteva, Normative virtual environments: Integrating physical and virtual under the one umbrella, in: Third International Conference on Software and Data Technologies (IC-Soft 2008), INSTICC, 2008, pp. 233–236.

[11] T. Trescak, Intelligent generation and control of interactive virtual worlds, Ph.D. thesis, Autonomous University of Barcelona, Barcelona, Spain (2012).

[12] P. Almajano, T. Trescak, M. Lopez-Sanchez, M. Esteva, I. Rodriguez, v-mwater: an e-government application forwater rights agreements, in: Agreement Technologies Handbook, Agreement Technologies, to appear in 2012.

[13] P. Almajano, T. Trescak, M. Lopez-Sanchez, M. Esteva, I. Rodriguez, An infrastructure for human inclusion in mas, in: ECAI '12, 2012.

[14] P. Almajano, T. Trescak, M. Esteva, I. Rodriguez, M. Lopez-Sanchez, v-mwater: a 3d virtual market for water rights (demonstration), in: AAMAS '12, 2012.

[15] A. Bourazeri, J. Pitt, P. Almajano, I. Rodrguez, M. Lopez-Sanchez, "meet the meter: Visualising smartgrids using self-organising electronic

institutions and serious games", in: SASO 2012, to appear in AWARENESS workshop at SASO 2012.

[16] P. Maes, D. Nardi (Eds.), Meta-Level Architectures and Reflection, Elsevier Science Inc., New York, NY, USA, 1988.

[17] S. Cranefield, G. Li, Monitoring social expectations in second life, in: Proceedings of the 5th international conference on Coordination, organizations, institutions, and norms in agent systems, COIN'09, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 133–146.

[18] S. Ranathunga, S. Cranefield, M. Purvis, Interfacing a cognitive agent platform with a virtual world: a case study using second life surangika ranathunga (extended abstract), in: Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 1181–1182.

[19] R. H. Bordini, M. Wooldridge, J. F. Hubner, Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology), John Wiley & Sons, 2007.

[20] G. B. Aranda, T. Trescak, M. Esteva, C. Carrascosa, Building quests for online games with virtual institutions., in: F. Dignum (Ed.), AGS, Vol. 6525 of Lecture Notes in Computer Science, Springer, 2010, pp. 192–206.

[21] I. G.-S. Sergio Alvarez-Napagao, Fernando Koch, J. Vazquez-Salceda, Making games alive: an organisational approach, in: Proceedings of AAMAS 2010 Workshop on Agents for Games and Simulations, 2010, pp. 112–124.

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

[23] M. Esteva, B. Rosell, J. A. Rodríguez-Aguilar, J. L. Arcos, Ameli: An agent-based middleware for electronic institutions, in: N. e. a. Jennings (Ed.), AAMAS 2004, Third international joint conference on au-

tonomous agents and multiagent systems, Vol. I, ACM, ACM, 2004, pp. 236–243.

[24] P. Noriega, Agent Mediated Auctions: The Fishmarket Metaphor, Monografies de l'Institut d'Investigació en Intel.ligència Artificial, Institut d'Investigació en Intel.ligència Artificial, 1999.