Dispatching Agents in Electronic Institutions

Hector G. Ceballos
Tecnologico de Monterrey
Ave. E. Garza Sada 2501
Monterrey, Mexico
ceballos@itesm.mx

Pablo Noriega
IIIA-CSIC
UAB Campus
Bellaterra, Spain
pablo@iiia.csic.es

Francisco J. Cantu
Tecnologico de Monterrey
Ave. E. Garza Sada 2501
Monterrey, Mexico
fcantu@itesm.mx

ABSTRACT

In Electronic Institutions [1], agents may be prevented from achieving their goals if other participants are not present in a given scene. In order to overcome this situation we propose the addition of an institutional agent in charge of dispatching agents to scenes through a participation request protocol. We further propose to endow this agent with the capability of instantiating new agents, thus providing grounds for a self-optimization of the system. Advantages of our proposal are illustrated with the implementation of an information auditing process.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

Keywords

Multiagent systems development environments. Electronic institutions. EIDE.

1. INTRODUCTION

The Electronic Institutions framework [1] developed in the Artificial Intelligence Institute of the Spanish National Scientific Research Council, (IIIA-CSIC), is a means to design and implement regulated open multiagent systems. The current framework is the outcome of more than a decade of developments and has been used to implement regulated MAS in several domains. For example, to support electronic auctions, to establish supply-chain virtual organizations, to agentify hotel and hospital management systems, to support participative experimentation, to model policy-making or to simulate human activity in archaeological sites. The framework has also served as a model-building environment for the discussion of topics like agent-based simulation, machine readable normative languages or autonomic computing. But in spite of this considerable variety of modes of use there have been very few published references to the underlying technology and, in particular, seldom any discussion of its expressive limitations and ways to circumvent them [2, 4].

This paper is one such discussion. We address the problem of deadlocks induced by improper institutional support in the follow-through of processes whose "most natural" representation may be as goal-directed workflows. In fact, we frame that problem in slightly more general terms: as the breakdowns produced by stalling or absent agents; and we advance a solution whose schema —an institutional agent with particular functionalities— may be reused *mutatis mutandis* to address similar problems and may also be coded as a standard functionality in the framework infrastructure. We believe that our solution should facilitate the adoption of the electronic institutions metaphor for the design of conventional MAS.

The structure of the paper is straightforward. The next subsections provide terminological and conceptual background. In Sec. 2 we present our proposal and in Sec 3 we describe a case study on information auditing to illustrate our proposal and present simple experimental results to back our claims. We finish with a brief discussion and comments on future work.

1.1 Electronic Institutions

For the purpose of this paper the EI framework may be described in terms of a conceptual model, a computational model and a software platform, EIDE, to specify and run electronic institutions [1, 5].

The conceptual model for electronic institutions assumes that the electronic institution determines a virtual space where agents interact subject to explicit conventions, so that institutional interactions and their effects *count as facts* in the real world. Because of this virtuality, it is assumed that all interactions within the electronic institution are *speech acts* expressed as illocutionary formulae. The electronic institution defines an *open MAS* in the sense that (i) it makes no assumption about the architecture and goals of participating agents (who may be human or software entities); and (ii) agents may enter and leave the institution at will, as long as the regimented conventions of the institution are met. Participating agents are subject to *role-based regulations* whose specification is given in terms of illocutions, norms and protocols. There are two classes of agents, internal and external. Internal agents act on behalf of the institution itself who is responsible for their behavior. External agents act on their own behalf and their internal state is inaccessible to the institution. Interactions are organized as repetitive activities called *scenes*. *Scenes* establish interaction protocols describing agent group meetings as transition diagrams whose arcs are labeled by valid illocutions. The *performative structure* captures the relationships among scenes describing those transitions agents playing certain role can make. Finally, *normative rules* describe the obligations an agent contracts while it participates in the institution. Agents may move from one scene to another, they may be active in more than one scene at a given time and they may perform different roles in different scenes.

The computational model for EIs defines a social (in-

stitutional) software layer between an agent communication platform (e.g. JADE) and participating agents. All *institutional communications* among agents are mediated by this platform. That institutional middleware is composed by three types of *infrastructure "agents"*: (i) An *institution manager* who centralizes valid communications and keeps track of the *state* of the institution, which is a data structure that contains the current values of all variables involved in the enactment of the institution. (ii) There are *scene* and *transition managers* for each, scene and transition, who handle the activation and persistence of scenes and transitions, and give access and exit to participants according to the local conventions; these managers mediate between the institution manager and the agent governors and keep track of the state of the institution as it applies to their particular context. (iii) One *governor* is attached to each agent and filters all communications between that agent and the institution; in particular, it directs valid illocutions to the corresponding scene managers and the institutional manager. The governor keeps a copy of the evolving state of the institution in order to apply regimented conventions on all speech acts its agent utters, and communicates to other infrastructure agents only those speech acts that comply with those conventions; thus the governor enables a change of the institutional state if and only if it admits a valid illocution from its agent.

The Electronic Institutions Development Environment, EIDE [5], consists of a graphical specification language, ISLANDER, whose output is an XML specification of an institution; a middleware AMELI [6], that takes an XML specification and enacts a runtime version of the institution with agents who run on a FIPA-compatible agent communication platform; a debugging and monitoring tool, SIMDEI, that registers all communications to and fro AMELI and displays and traces the evolution of the institutional state; finally an agent-shell builder, ABuilder, that from the XML specification produces an agent "skeleton" for each agent role. The skeleton satisfies all the (uninstantiated) navigation and communication requirements of the specification thus leaving the agent programmer to deal only with the implementation of the agent's decision-making logic at communication points.

The way these ideas are made operational in EIDE makes it possible to build complex regulated MAS. However, that operationalization corresponds more naturally to some types of MAS functionalities than to others. In this paper we propose one mechanism to deal with a type of situation that is currently difficult to handle in EIDE. Namely, the deadlock caused by the unavailability of an internal agent in a scene. Such deadlocks may happen in processes (performative structures) that require dedicated internal agents to follow-through subprocesses that involve single agents. For instance, the supervision of a patient's treatment through a medical protocol or, as exemplified in this paper, the auditing of the dossier of an individual as part of the academic evaluation process of a university. In general, this type of individual follow-through processes tends to appear whenever institutional interactions are organized in terms of individual agents' goals (e.g. each dossier has one agent who "pushes" the dossier through the steps of a pre-established workflow). Because in EIDE all agents that are present in a given scene share the state of the scene, when there is a process that involves private attention from the institution to one agent, an internal agent that enforces institutional conventions may be needed in all the scenes involved in the process. Moreover, currently there is no standard way of creating new internal agents in a running EI and furthermore, the current version of EIDE does not have the functionality of forcing an agent to act at any point, in particular, thus, it cannot force an agent to move from one scene to another, nor to terminate an agent. Consequently, deadlocks may arise when all available internal agents are already busy and also when no available internal agent enters the stalled scene.

To overcome this hurdle, the current solution in EIDE is to instantiate a new (sub) performative structure that corresponds to the private process each time that process needs to happen. In this solution, all the required internal agents are created automatically for every scene or transition in the process. However, the creation of substructures is expensive and in most cases involve having agents active in multiple scenes simultaneously, situation that is rather complex to program. Furthermore, this mechanism does not necessarily solve the deadlock induced by an available agent who stalls.

In this paper we propose another way of addressing that problem. It consists in the definition of a new internal *Dispatcher agent*, that keeps track of the need of internal agents (of those roles that may become necessary) and when requested by a scene manager, dispatches those that are available to the scenes that may need them, spawning new ones whenever necessary.

1.2 Agent Platform Services

Multi-agent frameworks have solved the problem of managing agent participation in several ways. For instance, FIPA has proposed a low-level solution for peer-to-peer communication based on services. On the other hand, multi-agent frameworks like Moise+ [8], MadKit [7] and Electronic Institutions have proposed upper-level solutions.[1]

The FIPA organization recommends a series of low-level services that any agent platform must implement to provide peer-to-peer communication: Agent Management System (AMS), Message Transport System (MTS) and Directory Facilitator (DF). The main role of an AMS is managing agent creation, deletion and migration on the agent platform, as well as maintaining the index of all agents identifiers in the platform. The MTS enables communication between agents internally or across different agent platforms. On the other hand, the DF functions as a yellow pages service on the agent platform. The DF considers agents as service providers and publishes the service descriptions provided by them. Even when the DF is an optional component of the agent platform, it is essential for finding the location of services in the multi-agent system. JADE and FIPA-OS are examples of frameworks that fulfill such recommendation.

In MOISE+ (ORA4MAS), organizations are modeled along three dimensions: structural, functional and deontic. Agents organize themselves in groups adopting roles that determine those missions they can or must perform in order to achieve the groupal goal. Social schemata allow to organize those missions and partial goals that must be performed/achieved

[1] A thorough review of how stalling and deadlocks are addressed in other platforms is beyond the scope of this paper, here we merely point towards the grounds that are common to most approaches and two environments that use a notion of interaction context assimilable to EI scenes.

in order to reach a common goal. Similarly, in AGR (Madkit) agents organize themselves in groups adopting one role and are limited to communicate only with other agents in the same group. Nevertheless, an agent can join multiple groups simultaneously. In Electronic Institutions, once inside a scene, agents are allowed to exchange messages only with other agents in that scene.

The three approaches propose a common space or context on which agents interact in order to achieve a single common goal, or multiple individual ones. Coordination requires controlling the entrance and exit of participants as well as the minimum/maximal number of agents playing certain role. Agents are free to decide which space(s) to join as their own goals dictate. However, none of the three approaches provides a solution for those cases when there are not enough agents for starting a group or scene, or when a group or scene is stalled waiting for one or more agents to continue. Agents in the stalled group need to communicate with other agents in order to invite them to join.

Our approach proposes the existence of a Dispatcher Agent in charge of directing the invitations made by agents in stalled groups/scenes. This agent optimizes the allocation of resources instantiating new agents when necessary and negotiating with available agents their participation.

2. REQUESTING AGENT PARTICIPATION

The context established by a scene in Electronic Institutions makes it difficult for agents in the scene to reach other agents. Such confinement creates a challenge: to warrant that all required agents be present in the scene. We formalize this problematic situation and propose a solution. We propose to institute an agent in charge of dispatching agents to scenes through a participation request protocol. This agent makes use of the low-level services recommended by FIPA and is coherent with the EI model and implementable in the current EIDE version.

2.1 Missing agents in scenes

Assuming that agents decide freely to enter or not in a scene, we may find two problematic situations in which participants would not achieve their individual goals: 1) not all the agents required for the scene are available, or 2) an agent in the institution is not aware that its participation is required in a particular scene. Both conditions can appear before the creation of the scene or during its execution. In the following we will only deal with the case of ongoing scenes.

Formally, the problematic situation would be defined as follows. There is an agent A_1 pursuing a goal G_1 that is currently playing role R_1 in scene S. Scene S is in state W_1 and the achievement of G_1 requires reaching state W_n. To do so, there is a sequence of illocutions $(M_1, ..., M_n)$ that must be issued by A_1 or by some other agent $(A_2, ..., A_n)$ playing roles $(R_2, ..., R_n)$ in S. Nevertheless, at state W_j the outgoing illocution M_j, $1 \leq j \leq n$, has for sender or receiver an agent playing role R_j for which there is no agent in the scene. We assume that there exists a state W_k, $1 \leq k \leq j$, at which the entrance of agents playing role R_j is allowed and in which A_1 is capable of keeping the scene on hold.

In order to reach W_n, agent A_1 sets $meansFor(G_3, G_2)$ and $meansFor(G_2, G_1)$, where $G_3 = \{holdAt(S, W_k)\}$ and $G_2 = \{agentsPlayingRole(R_j, S, Q)\}$. $meansFor(G_2, G_1)$ denotes that goal G_1 is in stand-by until G_2 is achieved. Likewise, $holdAt(S, W)$ is a goal that is satisfied when scene S reaches state W. Similarly, $agsPlayingRole(R, S, Q)$ is satisfied once there are Q agents playing role R on scene S, where the quantifier $Q \in \{\texttt{ONE}, \texttt{ALL}, \texttt{N=n}\}$, represents only one agent, any available agent, or exactly `n` agents, respectively.

In order to achieve the goal $agsPlayingRole(R, S, Q)$, the agent A_1 must request the participation of other agents through some protocol P. Such protocol P must achieve the agreement of Q agents to participate in S with role R. Protocol P might include agent selection and negotiation, that may be performed by A_1 itself or by another agent.

2.2 A Dispatcher Agent

We introduce the notion of *Dispatcher agent* as an intermediary agent that facilitates the achievement of those $agsPlayingRole(R, S, Q)$ goals owned by other agents. Let us represent the Dispatcher agent with the symbol A_D and denote its attributions with the role R_D. This agent keeps track of all agents on the institution through the *Agents* relation. Besides, A_D maintains the three following relations: *AgClasses*, *hasType* and *canPlay*. The set of agent classes $AgClasses = \{C_1, ..., C_n\}$ represent the software implementation of any participant, denoted by a source code class. Through $hasType \subset Agents \times AgClasses$, A_D keeps track of the agent class of every agent in the institution; it is assumed that every agent belongs to a single agent class. Finally, $canPlay \subset AgClasses \times Roles$ is used to know which roles may be played by an agent according to its agent class. The *canPlay* set may be built and updated by keeping track of participants in the institution, or may be known *a priori*.

A_D is capable of creating new instances of the agent class C_i through the action $Instantiate(C_i)$, which creates and enters an agent A_i in the institution. The configuration of A_D specifies, for each agent class, a maximum number of agents it can manage, denoted $MaxAgs(C_i)$. If $MaxAgs(C_i)$ is 0, A_D cannot instantiate agents of type C_i. The function $CurrAgs(C_i)$ counts the number of tuples $(AG_j, C_i) \in hasType$.

A_D implements two primitive operations for updating these relations. $RegisterAgent(A_i, C_i)$ inserts A_i in *Agents* and introduces the tuple (A_i, C_i) in *hasType*. On the other hand, $UnregisterAgent(A_i)$ removes A_i from *Agents* and the tuple (A_i, C_i) from *hasType*.

2.3 Processing Agent Participation Requests

We can assume that the dispatcher agent becomes aware of the $agsPlayingRole(R, S, Q)$ goal owned by the agent X through a protocol P_{Req}. In this way, A_D generates an *agent participation request* $APR(X, R, S, Q)$ whose purpose is committing Q agents to participate in S with role R. Given the previous definitions, A_D must determine if it can satisfy the request with the current set of agents.

Definition 1. An $apr = APR(X, R, S, Q)$ is *satisfiable w.r.t. Agents* if an only if, there is a set $AGS = \{AG_i | hasType(AG_i, C) \wedge canPlay(C, R)\ for\ 0 \leq i \leq m\}$, such that $m \geq 1$ for a quantifier $Q \in \{\texttt{ONE}, \texttt{ALL}\}$, or $m \geq n$ for a quantifier $Q = \texttt{N=n}$. Similarly, $APR(X, R, S, Q)$ is *unsatisfiable w.r.t. Agents* if $m = 0$ for $Q \in \{\texttt{ONE}, \texttt{ALL}\}$, or if $m < n$ for $Q = \texttt{N=n}$

Given the set of agents AGS that can satisfy *apr* and using a protocol P_{Inv}, agent A_D invites every agent in AGS

to participate in S playing role R.The acceptance of AG_i is represented by $Agree(AG_i, S, R)$, while its refusal is represented by $Refuse(AG_i, S, R)$. Once all agents in AGS have given a response, A_D proceeds to select the best agents for the scene.

The set of agents accepting the invitation, denoted $AccAgs$, is partially ordered by an operator $\succeq$ that calculates how suitable is an agent AG_i of type C for playing role R in scene S, denoted $AccAgs_{\succeq}$. Hence, $AG_i \succeq AG_j$ means that AG_i is better or at least as good as AG_j for the given scenario.

The ordered set $SelAgs \subseteq AccAgs_{\succeq}$ is constituted by the first n agents of $AccAgs_{\succeq}$, where $n = 1$ for $q =$`ONE`, $n = |AGS|$ for $q =$`ALL`, and $n \leq$ `n` for $Q =$ `N=n`.

Definition 2. An $apr = APR(X, R, S, Q)$ *is satisfied*, denoted $satisfied(apr)$, if $|SelAgs(apr)| \geq 1$ for $Q \in \{$`ONE`, `ALL`$\}$ or $|SelAgs(apr)| =$ `n` for $Q =$ `N=n`.

If apr is not satisfied w.r.t. $Agents$, the introduction of new agents in the system would solve the problem. This is possible if there exists at least one agent class C_i such that $canPlay(C_i, R)$ and $MaxAgs(C_i) > 0$. If there is no C_i with these properties, A_D will have to wait for new agents for a fixed period of time τ, after which it will declare the request *unsatisfied*.

Given that there may be more than one agent class capable of playing role R, A_D can use the same partial order criteria $\succeq$ for selecting the best class for the role R required in an apr. If apr is an agent participation request and $AgClss(apr)$ a partially ordered set $\{C_i | canPlay(C_i, R)\}$ w.r.t. $\succeq$, then A_D will choose the first $C_i \in AgClss(apr)$ for which $CurrAgs(C_i) < MaxAgs(C_i)$. If no C_i satisfies this requisite, the instantiation is not performed.

A_D determines the number of agents that should be instantiated to satisfy the request, denoted $NMissing$. If $Q =$ `N=n`, $NMissing = n - |SelAgs|$, meanwhile if $Q \in \{$`ONE`, `ALL`$\}$ then $NMissing = 1$. If $NMissing > 0$, A_D can instantiate and enter in the institution a *missing agent* through the execution of the primitive $Instantiate(C_i) : A_i$ for some $C_i \in AgClss(apr)$. These primitives make use of the Agent Management System (AMS) provided by any FIPA-compliant agent platform.

Agents entering the institution are invited to scenes held in stand-by due to an unsatisfied apr. Thus, if an agent AG_i of type C_i is created by A_D in order to satisfy $apr = APR(X, R, S, Q)$ and AG_i doesn't accept the corresponding invitation to S, C_i is removed from $AgClss(apr)$. If an agent created by A_D refuses all the invitations made during its logging in the institution, its access is denied. Agents exiting from the institution produce a revision of unsatisfied agent participation requests that might require the instantiation of new agents.

We distinguish between permanent and transient participants according to their patterns of entry and exit in the institution. Let's call *permanent participants* those agents that remain in the institution continuously while it is alive. On the other hand, *transient participants* are agents that enter the institution pursuing certain goals and exit once they have reached them. Agent classes representing permanent participants are identified by the set $PermAgCls \subset AgClasses$; similarly transient participants are denoted by $TranAgCls \subseteq AgClasses$.

Now we can establish necessary conditions to determine when an unsatisfied agent participation request justifies an agent instantiation.

THEOREM 1. *An unsatisfied $apr = APR(X, R, S, Q)$ can be satisfied through the instantiation of $NMissing$ agents if there is a subset $(C_i \cup C_j) \subseteq AgClss(apr)$ such that $NMissing \leq FSlots(apr)$, where*

$$FSlots(apr) = \sum_i MaxAgs(C_i) - CurrAgs(C_i, S) + \sum_j MaxAgs(C_j) - CurrAgs(C_j)$$

for $C_i \in (AgClss(apr) \cap TranAgCls)$ and $C_j \in (AgClss(apr) \cap PermAgCls)$. $CurrAgs(C_i, S)$ returns the number of agents with type C_i currently in scene S.

PROOF. Eventually, transient agents will leave the institution releasing slots that A_D can use for creating new instances, hence in the worst case where $MaxAgs(C) = CurrAgs(C)$ and a single $C \in AgClss(apr) \cap TranAgCls$ exists, the exit of all agents of type C will make $CurrAgs(C) = 0$ allowing the instantiation of the required agents.

$CurrAgs(C, S)$ allows to consider those agents of class C that will remain in S. For permanent agents, we cannot assume that they will exit from the institution, hence we can only count with the instantiation of $MaxAgs(C) - CurrAgs(C)$ agents of type C. □

The order in which invitations are issued is important when incoming agents have a limited capacity for attending invitations. Suppose that A_D is processing two agent participation requests apr_1 and apr_2 for the same role R, where $AgClss(apr_1) = AgClss(apr_2)$, and the maximum number of invitations an agent of type $C \in AgClss(apr_1)$ can take is one. If an agent is instantiated for class C and the invitation for apr_2 is sent earlier than the invitation for apr_1, the new agent will only attend the scene in apr_2. Similar instantiations and invitations might satisfy apr_2 and left apr_1 in hold if $NMissing_{apr_1} < FSlots(apr_2)$.

On the other hand, the refusal of agents for participating in an apr might produce an empty $AgClss(apr)$ set. This condition would allow A_D to consider apr unsatisfiable discarding it from its queue. Otherwise, an unsatisfiable apr_1 might block a subsequent apr_2 if $AgClss(apr_2) \subseteq AgClss(apr_1)$.

2.4 Request and Invitation Protocols

Agent participation is negotiated through two protocols, one for requesting agent participation (P_{Req}) and another for inviting agents (P_{Inv}). Both protocols must be executed in parallel with the scene that originated the request for agent participation.

Let us use the *DAgent* name for denoting the R_D role, call *ReqAgent* the role played by an agent requesting the participation of other agents and call *InvAgent* the role that an invited agent plays. Every agent in the institution must be able to play *ReqAgent* and *InvAgent* roles, meanwhile only one agent, A_D, is allowed to play the role *DAgent*.

Figure 1 shows the sequence diagram for P_{Req} between *DAgent* and *ReqAgent*. Figure 2 depicts the automata describing the request protocol where letters on arrows represent valid sequences of illocutions taken from figure 1, as well as the nested call to P_{Inv}.

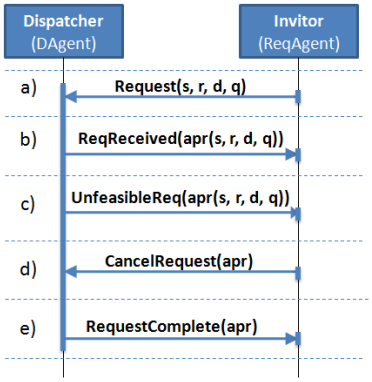

Figure 1: Sequence diagram for the request protocol.

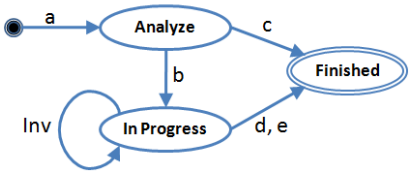

Figure 2: Automata for the request protocol.

Figure 3 shows a sequence diagram with segments of P_{Inv}. Figure 4 illustrates the automata describing the request protocol where letters on the arrows represent sequences of illocutions shown in Figure 3. P_{Inv} distinguishes between invitations to current agents and the instantiation and invitation of new agents. Figures are discussed below.

3. CASE STUDY

We used the Electronic Institution formalism for the automation of the auditing of an information repository. Several kinds of autonomous agents and human users participate in this auditing process. The process is initiated by external events and during its execution human intervention might be required. Rather than waiting for human users entry to the system, our approach enables autonomous agents to request human participation in order to achieve their goals.

A multiagent system for performing this auditing process was implemented with the tools developed in the IIIA [5]. Experiments and the results obtained are described at the end of this section.

3.1 Information Auditing

The information repository is managed by a *RepGuardian* agent that monitors changes on the repository and initiates the auditing process. The auditing process is driven by a specialized agent *Carrier*, and with the participation of other autonomous agents, *Auditor* and *Corrector*, as well as user agents representing human *experts* and information *authors*. A Carrier agent receives a notification about a record that has been added or modified in the information repository. The Carrier requests every available Auditor agent to check the internal consistency of the repository with respect to the

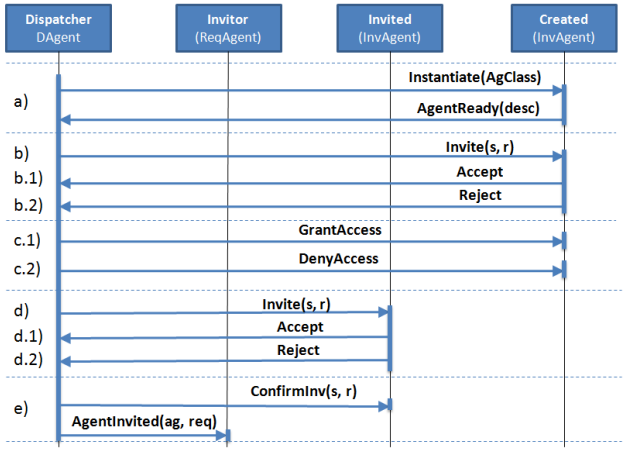

Figure 3: Sequence diagram for the invitation protocol.

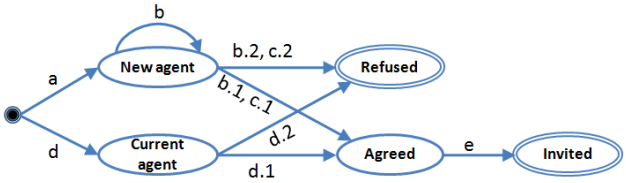

Figure 4: Automata for the invitation protocol.

auditing rules it knows. The Auditor agent responds to the Carrier whether informing that the record and the repository are consistent or returning a set of the inconsistencies detected. The type of inconsistencies are either internal inconsistencies of the record, or violations of rules defined for the entire repository; for instance, duplicity of records.

The Carrier agent chooses between sending the record to automatic correction with a Corrector agent, asking for expert assessment from a human expert, or notifying the author of the record of the possible inconsistency. The Corrector agent can apply the correction procedure or ask for expert assessment instead. In turn, the Expert user can modify the record or notify to its author. At the end, the decision made by the author is final. This decision model is depicted on Figure 5.

In this scenario we can detect some cases where human

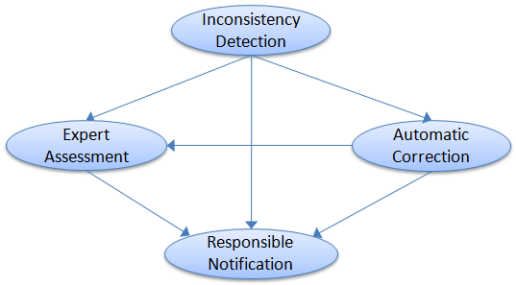

Figure 5: Decision model during information correction.

users are not present in the system when their participation is required. For instance, the user registering or modifying the record might have left the system by the time an inconsistency needs a final decision. Similarly, the expert user may not be logged in the system when an inconsistency is detected. A new auditing rule evaluated throughout the repository might require the assessment of expert users or users responsible for inconsistent records.

3.2 Implementing the auditing process

The process described above was specified with a performative structure *AuditingPS* that contains protocols for: triggering the auditing (NewInfo), detecting inconsistencies (Auditing), performing the corrections (Correction) and integrating results on the repository (Audited). *AuditingPS* and its protocols use the roles defined above: RepGuardian, Carrier, Auditor, Corrector, Expert and Author. Using the ABuilder tool [5], agent classes were generated for each role, except for Expert and Author roles which shared the same agent class, named *UserAgent*. Only the *RepGuardianAgent* was classified as permanent; the rest of the agent classes were considered transient.

It was possible to simulate the auditing of new pieces of information registered in the repository with this implementation. The simulation required to have a fixed set of user agents playing the roles of experts and authors for every possible human user that could be required in the process. Besides, every expert or author user participated in each Correction scene.

Given that human users responses to a request made when he/she was off-line might take entire days, the duration of Correction scenes was limited by a timeout after which the correction is considered to have failed. A User agent representing an expert or an author is not allowed to participate on multiple scenes simultaneously; nevertheless it can accept invitations to other scenes until reaching a given limit of invitations.

3.3 Implementing Agent Participation Request

The request for agent participation was implemented by developing: 1) an additional performative structure containing the protocols proposed in our approach, 2) the dispatcher agent, 3) an institutional service for instantiating new agents, and 4) new functionality for previously defined agent classes.

The new performative structure is assembled with four protocols that enable agents to: 1) log in, 2) request agent participation, 3) receive and answer invitations to scenes, and 4) log out. *AuditingPS* was inserted in this performative structure indicating that every agent should pass by the first three protocols before entering *AuditingPS*, hence remaining active in request and invitation protocols. Finally, after leaving *AuditingPS* they should pass by the log out scene. Protocols and roles specified in this performative structure are defined in section 2.4.

The *RepGuardian* agent implemented the functionality of the *DAgent* role with the characteristics described in section 2.2, the algorithm outlined on section 2.3 and the primitives described in both sections.

Agents developed for *AuditingPS* were augmented with the functionality of *ReqAgent* and *InvAgent* roles. A parameter on each agent class C denoted $MaxInv(C)$ was set to limit the maximum number of simultaneous invitations an agent of this class can accept.

Parameter	Low	High	Critical
Feeding rate	40 sec.	10 sec.	10 sec.
Expert revision	2-5 sec.	2-5 sec.	2-5 sec.
Author revision	20-25 sec.	20-25 sec.	20-25 sec.
MaxAgs(Carrier)	10	10	10
MaxAgs(User)	10	10	5

Table 1: Experiment configurations.

3.4 Experiments

To demonstrate the capabilities of the Dispatcher agent we prepared a test-bed with the system described above. We want to observe the capabilities of the *DAgent* for dispatching agents to scenes where human intervention is requested and for detecting unsatisfiable requests. In order to do so, we simulated different demand patterns on the system and manipulated the maximum number of agents permitted. An overloaded system is that in which information is fed faster than users are able to revise it. Thus we provoked that certain scenes stalled due to the lack of enough agents for all of them. Next we manipulated the maximal number of agents in order to generate unsatisfiable requests.

We defined a single RepGuardian and constant populations of auditor and corrector agents. One Carrier agent was instantiated for each information piece fed into the system. User agents playing the role of Expert or Author are created on demand up to a maximum of $MaxAgs(User)$. The same User agent representing a human user must participate in all the scenes where the user intervention is requested and it must wait to finish its work in a scene before proceeding to the next.

Our focus was on the Correction protocol, whose decision model is shown in Figure 5 and is explained in section 3.2. In our experiments, the power for instantiating Corrector agents was disabled, i.e. $MaxAgs(Corrector) = 0$. In consequence, the dispatcher informs the Carrier of the unfeasibility of requests for Corrector agents. Hence the Carrier requests an expert who in turn calls one author for correcting the record. In conclusion, every Correction scene is initiated by one Carrier and requires the participation of one expert and one author. All the agents remain in the scene until this finishes. User agents were limited to accept up to three invitations, i.e. $MaxInv(User) = 3$.

We prepared three system configurations. The first configuration gives us a reference of how the system would behave under low demand. In the second we have an information feeding rate higher than revision time, which we expect to generate several stalled scenes. And the last configuration has a reduced number of User agents for detecting unsatisfiability of requests. Parameters for the three configurations, labeled Low, High and Critical respectively, are shown in Table 1.

3.5 Results

Using the configurations given above we ran experiment rounds auditing 50 new information pieces in order to measure the behavior of agents and measure the performance in the Correction scene. We observed the maximal number of simultaneously stalled scenes, i.e. scenes in hold due to a request for agents, and calculated the average conclusion time for these scenes. Aditionaly, we observed the maximum

Observation	Low	High	Critical
Max. stalled scenes	2	7	10
Avg. scene time	30 sec.	61 sec.	197 sec.
Max. active Carriers	2	10	10
Max. active Users	2	10	5
Max. active Experts	1	5	4
Max. active Authors	1	6	2

Table 2: Experimental results.

number of concurrent Carrier and Users agents, as well as the maximum simultaneous number of User agents playing the Expert or Author role; recall that experts and authors use the User agent class for participating in the system. Results for the three configurations are shown in Table 2.

The first configuration showed only one Correction scene most of the time, and reached the maximum of two at some point of the simulation. The number of simultaneously stalled Correction scenes and Carrier agents is the same as long as Carrier agents are in charge of creating the Correction scene. Only one expert and one author were active in the system at the same time, authors and expert were released once the correction scene finished and were instantiated again when a new Correction scene was generated.

In the second configuration the reduction on the feeding rate produced more stalled scenes and a higher utilization of agents. The maximal number of Carrier and User agents was reached. This time the maximum number of stalled scenes didn't match the maximal number of Carriers agents because they were busy participating in other scenes of the auditing process. The average conclusion time for scenes was doubled as long as busy experts kept in hold at most two scenes meanwhile they were attending another scene. Figure 6 shows the behavior of the population of Carrier and User agents, broke down on Expert and Author roles, for this configuration.

In the third configuration, after approximately twelve successful evaluations the entire system stalled. At this point we observed the five user agents playing the role of Expert leaving no space for authors. Ten scenes were stalled, five of which had an Expert agent and the other five had a single Carrier agent waiting. In this case the dispatcher agent was not capable of determining the unsatisfiability of the requests for authors as long as it was expecting that some User agent left the institution for instantiating an agent to play the author role. The rest of the scenes made use of the five available Expert agents not allowing the instantiation of a new User agent for playing the role of Author. All these scenes finished thanks to the timeout of 200 seconds, as can be observed in the average termination time for these scenes.

This last scenario make us conclude that it was necessary to reserve agent slots for authors in order to conclude the scenes satisfactorily. Even when the participation of experts and authors is not assured in all the scenes, we should be capable of indicating it to the dispatcher agent in order to prevent the deadlock.

4. DISCUSSION

Low-level services like the Directory Facilitator only answer questions about the current set of agents in the system. On agent platforms implementing this kind of services an agent must search agents in term of the services they can provide. For Electronic Institutions such service could represent playing a role at certain type of scene. Nevertheless, the agent should be capable of negotiating the participation of other agents directly with them. In our approach this negotiation is centralized and organized in the $DAgent$ which allows to detect unsatisfiable requests at some extent.

Another advantage of our approach is that populations of agents can be adjusted on line according to the current demand. This is possible thanks to the ability of transient agents for leaving the institution when they are idle and to the $DAgent$'s ability for instantiating agents when they are required.

Another way of optimizing the system performance is using a well known protocol for resource allocation, the ContractNet protocol [9]. ContractNet can be adapted for being used as agent request protocol. This can be done by narrowing the signal task announcement to agents of class $C \in AgClss(apr)$ and making an analogy between *invitation acceptance* and *task assignment*, where the task abstraction would be expressed as $playRoleIn(AG, R, S)$. The bid specification can include information about the time that would take a bidder to get to the scene. Finally, every agent AG_i chosen from $AccAgs(apr)$ receives an AWARD message for $playRoleIn(AG, R, S)$, and AG_i is added to $SelAgs(apr)$. Agents making a bid for a task of this type commits since that moment to attend to the scene if it is awarded.

5. CONCLUSIONS

We presented an approach for facilitating goal achievement by agents on an Electronic Institution. The type of situations prevented are those where a missing agent prevents the on-going execution of a protocol. Our approach consists in introducing an agent that dispatches available or new agents to those scenes.

We proposed necessary conditions for the instantiation of agents to satisfy an agent participation request. Nevertheless, experiments showed that such conditions are not sufficient when the scene requires the simultaneous participation of further agents of the same class. More work on this direction is needed. Even though, agent instantiation controlled by the $DAgent$ showed its potential for optimizing dynamically the populations of agents in the system.

Advantages of our approach were illustrated in an auditing scenario with particular characteristics. For example, the interaction was not initiated by human users but by autonomous agents. As it was shown, our approach allowed the participation of just the necessary agents on each scene and avoided having idle agents in the system.

5.1 Future Work

The request protocol can be extended to deal with future agent invitation and not just current invitations. By so doing, we could prevent the deadlock of concurrent scenes. Another option would be developing an algorithm for pruning the directed cyclic graph representing an EI protocol or scene. That would produce a reduced version of the protocol when agents for certain role are missing. A pruned protocol that doesn't reach the final state would indicate an unsatisfiable scene execution. For instance, a Correction protocol on which the participation of Corrector, Expert and Author agents is pruned, could be detected *a priori* as unsuccessful.

An institutional model of public information for scenes and agents can be used to improve our proposal. Public

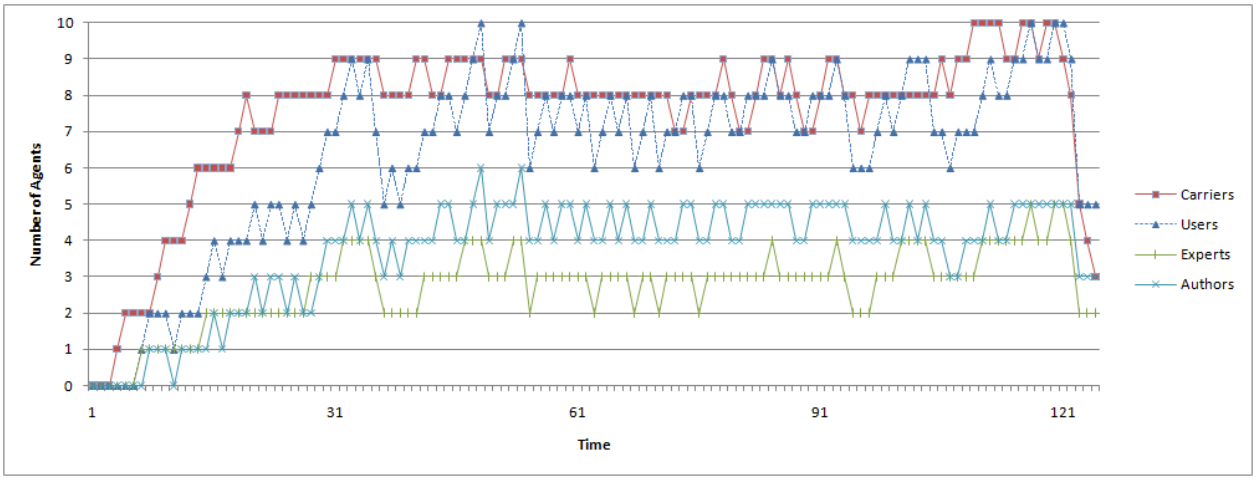

Figure 6: Agent populations on the high-demand configuration.

information of the scene and the participants may be used by the *DAgent* to narrow the announcement task, and by invited agents to calculate their bids for participating in a scene. For example, a Corrector agent that knows a rule for correcting inconsistencies of a single type, should be directed only to scenes where an inconsistency of that type is being corrected.

Agent descriptions formalized through a Description Logics [3] system would allow the generation of agent profiles describing the properties that potential participant agents should have. For instance, knowing that there are three auditing rules for the repository and that every Auditor agent can only handle one single rule, the request for auditor agents for all the type of auditing rules would generate three agent profiles, one for each rule. An instance of each profile would be enough for assuring a complete auditing of each new record.

6. ACKNOWLEDGMENTS

This paper was partially funded by the Spanish Ministry of Science and Innovation AT (CSD2007-0022, INGENIO 2010) and EVE (TIN2009-14702-C02-01), by the Generalitat de Catalunya 2009-SGR-1434, by the Mexican Council for Science and Technology and by the Tecnologico de Monterrey.

7. REFERENCES

[1] J. Arcos, M. Esteva, P. Noriega, J. Rodriguez-Aguilar, and C. Sierra. Engineering open environments with electronic institutions. *Engineering Applications of Artificial Intelligence*, (18):191–204, March 2005.

[2] J. L. Arcos, P. Noriega, J. A. Rodríguez-Aguilar, and C. Sierra. *E4Mas through Electronic Institutions.*, pages 184–202. Number 4389. Springer, Berlin / Heidelberg, 08/05/2006 2007.

[3] F. Baader. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, September 2007.

[4] J. Campos, M. López-Sánchez, J. A. Rodríguez-Aguilar, and M. Esteva. *Formalising situatedness and adaptation in Electronic Institutions.*, volume LNCS 5428, pages 126–139. Springer-Verlag, 2009.

[5] M. Esteva, J. A. Rodríguez-Aguilar, J. L. Arcos, C. Sierra, P. Noriega, and B. Rosell. Electronic institutions development environment. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems. AAMAS-08*, pages 1657–1658, Estoril, Portugal, 12/05/2008 2008. International Foundation for Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems.

[6] M. Esteva, J. A. Rodríguez-Aguilar, B. Rosell, and J. L. Arcos. Ameli: An agent-based middleware for electronic institutions. In *Third International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS'04)*, New York, USA, July 19-23 2004.

[7] O. Gutknecht and J. Ferber. MadKit: Organizing heterogeneity with groups in a platform for multiple multi-agent systems. Technical Report R.R.LIRMM 9718, LIRM, December 1997.

[8] R. Kitio, O. Boissier, J. F. Hubner, and A. Ricci. Organisational artifacts and agents for open multi-agent organisations. In *Coordination, Organizations, Institutions, and Norms in Agent Systems III*, pages 171–186, 2008.

[9] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12), December 1980.