Requesting agent participation in Electronic Institutions (Extended Abstract)

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

The Electronic Institutions (EIs) framework is designed for regulating interactions among heterogeneous agents in open systems [1]. In EIs, agent interactions are speech acts whose exchange is organized as conversation protocols called *scenes*. Agents can participate simultaneously in multiple scenes playing a single role in each one of them. However, at some point, the execution of a given scene may require the presence of an agent playing a particular role. When such an agent is missing, a deadlock may ensue unless the institution or the agents themselves can invoke the participation of an agent to play the missing role. Such functionality is not provided in the current EI framework. We propose an extension of the framework that addresses that problem in a generic way: the provision of an institutional agent in charge of instantiating new agents and dispatching them to scenes through a participation request protocol. In this paper we make the proposal precise and illustrate it with a use case.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Agent-oriented software engineering. Development environments. Electronic institutions.

1. REQUESTING AGENT PARTICIPATION

A *Dispatcher agent* is as an intermediary agent that facilitates recruiting agents to participate in a scene. Let us represent the Dispatcher agent with the symbol A_D and denote its attributes with its $DAgent$ role. A_D keeps track of all agents in the institution through the relation $Agents = \{A_1, ..., A_n\}$. Additionally, A_D is involved in three relations: $AgClasses$, $hasType$ and $canPlay$. The set of agent classes $AgClasses = \{C_1, ..., C_n\}$ is an equivalence relation that represents 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. Finally, $canPlay \subset AgClasses \times Roles$ denotes the roles that may be played by an agent according to its agent class.

A_D is endowed with three primitives, one for creating new agent instances and two more for maintaining the previously introduced relations. The $Instantiate(C_i)$ primitive creates a new instance A_i of the agent class C_i and enters it in the institution. This primitive makes use of the Agent Management System (AMS) provided by any FIPA-compliant agent platform. The configuration of A_D specifies, for each agent class, the maximum number of agents it can handle, denoted $MaxAgs(C_i)$. The $RegisterAgent(A_i, C_i)$ primitive inserts A_i in $Agents$ and introduces the tuple (A_i, C_i) in $hasType$. Similarly, the $UnregisterAgent(A_i)$ primitive removes A_i from $Agents$ and the tuple (A_i, C_i) from $hasType$. The function $CurrAgs(C_i)$ returns the number of tuples $(A_j, C_i) \in hasType$.

1.1 Request and Invitation Protocols

An agent A_1 playing role R_1 in scene S may be unable to achieve a goal if another agent playing role R is missing in that scene. Let $agsPlayingRole(R, S, Q)$ denote A_1's requirement of Q agents playing role R in scene S, where the quantifier $Q \in \{\texttt{ONE}, \texttt{ALL}, \texttt{N=n}\}$, represents only one agent, all available agents, or exactly `n` agents, respectively. Agent participation is negotiated through two protocols, one for requesting agent participation (P_{Req}) and another for inviting agents to scenes (P_{Inv}).

Through P_{Req}, A_1 (playing role $ReqAgent$) informs A_D of its request for agent participation. A_D processes the request and informs A_1 whether enough agents accepted its invitation or that its request was unsatisfied.

Through P_{Inv}, A_D sends invitations to available or new agents (playing role $InvAgent$), who may accept or decline the invitation. A new agent A_i is invited to those scenes that motivated its instantiation before entering the institution and denied access, by A_D, if it doesn't accept any of these invitations.

1.2 Processing Agent Participation Requests

In P_{Req}, A_D generates an *agent participation request* that identifies the request from agent A_R for committing Q agents to participate in S with role R, denoted $APR(A_R, R, S, Q)$. First, A_D determines *satisfiability* of *apr* based on the availability of agents capable of playing role R. In order to nar-

Cite as: Requesting agent participation in Electronic Institutions (Extended Abstract), Hector Ceballos, Pablo Noriega and Francisco Cantu, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 1375-1376

row the invitation to agents capable of playing role R, the set of agent classes to consider is given by $AgClss(apr) = \{C_i | canPlay(C_i, R)\}$. Potential availability for apr is given by $Cap(apr) = \sum_i MaxAgs(C_i)$ for each $C_i \in AgClss(apr)$.

Definition 1. An $apr = APR(A_R, R, S, Q)$ is *satisfiable* if $Cap(apr) \geq 1$ for a quantifier $Q \in \{\texttt{ONE}, \texttt{ALL}\}$, or $Cap(apr) \geq n$ for a quantifier $Q = \texttt{N=n}$. Otherwise, apr is considered *unsatisfiable*.

Unsatisfiable requests are canceled and notified to the requester. A_D continues processing feasible requests identifying the set of available agents for apr, $Avail(apr) = \{A_j | Agents(A_j) \wedge hasType(A_j, C_i) \wedge canPlay(C_i, R)\}$. Next, A_D progressively invites agents $A_j \in Avail(apr)$ until reaching the quota of accepted invitations or until finishing with the list. The list of agents accepting the invitation for apr is denoted $AccAgs(apr)$ and is used for determining apr's satisfaction.

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

If apr is not satisfied with the current set of agents, A_D will try to instantiate and invite additional agents of type $C_i \in AgClss(apr)$ such that $MaxAgs(C_i) - CurrAgs(C_i) > 0$. If there is no capacity for agents of type C_i, A_D will wait for agents becoming available or for slots released by agents leaving the institution. After a fixed period of time A_D will declare the request *unsatisfied*. Both satisfied and unsatisfied requests are notified as completed to A_R in P_{Req}.

2. CASE STUDY

We used the the software platform EIDE [2] for implementing an information auditing process. We generated multiple Correction scenes on which a software agent may be required to perform the automatic correction (Corrector role) of a record, a human auditor may be required to supervise the correction (Expert role) and the corresponding author of the record may be required to make a final assessment (Author role). In each case, the participation of these agents is not assured and record's information determines which users must be addressed.

2.1 Implementation

Our proposal was incorporated in an initial system implementation through the introduction of a new performative structure with four scenes where agents: 1) log in, 2) request agent participation, 3) receive and answer invitations to scenes, and 4) log out. A_D is permanently present in the four scenes. The auditing process modeled in the original performative structure, *AuditingPS*, is nested in this performative structure making every agent pass through the log-in scene and remain active in the request and invitation scenes. After leaving *AuditingPS*, agents pass through the log-out scene. The functionality of the Dispatcher agent was implemented in one of the original agents. Agent instantiation was implemented through a new institutional service. Agents originally developed for *AuditingPS* were augmented with the functionality of *ReqAgent* and *InvAgent* roles.

2.2 Experiments and Results

For testing purposes we set up three system configurations. The first configuration corresponds to a low demand setting. The second had an information feeding rate higher than revision time. The last configuration had a reduced number of User agents hence generating unsatisfiable requests.

As expected, in the first configuration we only had a single scene at a time and one agent playing each role in the system. In the second configuration the reduction on the feeding rate produced stalled scenes and a higher utilization of agents; the maximal number of User agents was reached but stalled scenes were released after the exit of busy agents. Finally, in the third configuration a deadlock was produced when all user agents were busy and a second request for user agents in stalled scenes was unsatisfied. This situation could be prevented by warranting the availability of agents for the scene before issuing the first agent request.

3. DISCUSSION

Our approach enables mediated scene coordination and proposes agent instantiation as a mechanism for regulating agent populations online. The participation of agents is negotiated centrally by the Dispatcher agent which allows to detect unsatisfiable requests to some extent. Experiments show the necessity of extending our approach to handle requests for *future* participation, enabling agent slot reservation and release, at the start and at the end of a scene, respectively. This approach can be applied on Madkit [3] and ORA4MAS [4], where coordination in groups shows the same limitation that EI scenes.

4. 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.

5. 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] 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*.

[3] 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.

[4] 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.