Java Interagents for Multi-Agent Systems

Francisco J. Martín, Enric Plaza, Juan A. Rodríguez-Aguilar, and **Jordi Sabater**
IIIA - Artificial Intelligence Research Institute CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Barcelona, Spain
{martin,enric,jar,jsabater}@iiia.csic.es

Abstract

In this paper we introduce an *interagent* as an autonomous software agent which manages (intermediates) the communication and coordination between an agent and the agent society wherein this is situated. With this aim, we have developed JIM, a general-purpose interagent that provides agents with a highly versatile range of programmable —before and during the agent's run-time— communication and coordination services.

Introduction

There exists a number of problems which involve multiple sources of knowledge and, thereby, can best be addressed using a multi-agent system (MAS) — a computational system composed of several interacting agents which cooperate with one another to solve complex tasks. Furthermore, the deployment of multi-agent systems permits to benefit from a number of advantages —such as parallelism, robustness and scalability— that a single agent working isolatedly can not offer itself. Coordinating the activities of the several agents composing a MAS is essential in order to guarantee the proper workings of a MAS itself and benefit from such advantages.

Currently, we are partners of the SMASH project[1], a collective, joint effort involving several research institutions that addresses the construction of multi-agent systems that help solve problems of distributed nature in hospital services. The development of such multi-agent systems requires the deployment of (highly flexible) communication and coordination mechanisms to integrate a set of heterogeneous agents —agents developed by different people for different purposes and in different languages— within a common setting. Instead of letting agents deal themselves with such issues, our proposal opts for introducing an autonomous software agent that we call *interagent* which manages (intermediates) the communication and coordination between the agent it is attached to and the agent society wherein it is situated.

[1] http://www.iiia.csic.es/Projects/smash/

Interagents

In our proposal, the functionality provided by an interagent will highly depend on the role played by the agent interacting with it. Thus we distinguish two distinct roles for agents making use of interagents: *i)* the *user* of an interagent regards it as the sole and exclusive means through which it can interact with the agent society thanks to the set of communication and coordination services provided by the interagent, but previously defined by the *owner*; *ii)* the *owner* of an interagent is provided with a wide range of facilities to either load or program into the interagent the communication and coordination services that the user is allowed to employ. Needless to say, an agent can possibly play both roles at the same time.

Interagents —like KQML facilitators (Patil *et al.*, 1992)— are inspired by the *efficient secretary* metaphor already introduced in the Actors model of concurrent computation. Interagents —unlike KQML routers— offer the coordination level required by agents to cooperate in non-trivial ways. Basically, an interagent is a component which supports a dynamically programmable level of interaction.

We have developed JIM, a general-purpose interagent that provides agents with a highly versatile range of programmable —before and during the agent's run-time— communication and coordination services[2].

Communication Services

An interagent is informed by its user about the message to be sent and its addressee, and then the interagent carries out all the operations needed to deliver it correctly. An interagent and its user can communicate in

[2] A full paper on JIM is available at http://www.iiia.csic.es/Projects/fishmarket/publications-team.html

two ways: *a)* through (TCP-)stream-sockets —in case that an interagent and its user are two distinct computational processes (residing in the same computer or not); *b)* through shared memory —in case that an interagent and its user are two distinct threads residing in the same process space.

An interagent provides its user with the following communication services based upon TCP/IP: *i)* queueing and serialization of outgoing messages from its user and queueing and serialization of incoming messages from (the interagents of) other agents; *ii)* asynchronous communication between agents; *iii)* synchronous communication between agents (implemented on top of buffered asynchronous communication between interagents); *iv)* agent naming services (white pages); *v)* handling of expired messages and automatic recovery of transmission errors.

Coordination Services

An interagent allows interdependencies between agents' communicative acts, expressed as performatives of high-level agent communication language, to be ordered by means of *conversation protocols* which represent the conventions adopted by agents when interacting through the exchange of messages.

We have modeled and implemented conversation protocols as a special type of pushdown automata because unlike finite state machines, pushdown automata allow to store and subsequently retrieve the context of an ongoing conversation. On the one hand, each state in the finite state control of the pushdown automaton represents the situation of an agent during an ongoing conversation. On the other hand, each transition in the automaton indicates what message has to be either sent or received to produce a transition in the conversation protocol. Therefore, it can be said that interagents constrain what an agent can utter and hear, and when.

An interagent can support a wide range of conversation protocols that can be declaratively defined *statically* (before the user's run-time, as an element to be stored in the library of conversation protocols) and *dynamically* (the owner can interactively define new conversation protocols at run-time using a conversation protocol definition language). This capability of allowing agents to alter and define themselves their conversation protocols at run-time distinguishes them from other approaches like COOL (Barbuceanu and Fox, 1995) or JAFMAS (Chauhan, 1997).

Conclusions and Ongoing Work

An interagent provides an agent with the basic mechanisms to interact (communicate and coordinate) with other members of an agent society. In this way, the overload related to the management of the communication and coordination tasks needed by an agent to live in a multi-agent system is shifted to its interagent, that relieves its user from such a "tedious" work.

Two major benefits are gained from employing interagents. On the one hand, it permits agents to reason about both communication and coordination at a higher level of abstraction, whereas on the other hand it provides a complete set of facilities that allows agent engineers to concentrate on the design of their agents' inner and social behavior.

JIM is currently being used in two directions: *i)* to promote the knowledge representation language Noos to an agent-oriented language (Martin *et al.*, 1998); *ii)* to coordinate the activities of the market intermediaries composing the *Fishmarket*[3] system (Rodriguez-Aguilar *et al.*, 1998) and the interaction between the market as a whole and the participating buyers and sellers.

Acknowledgments This work has been supported by the Spanish CICYT project SMASH, TIC96-1038-C04001 and the DGR-CIRIT doctoral scholarships FI-PG/96-8490 and FI-DT/96-8472.

References

Mihai Barbuceanu and Mark S. Fox. Cool: A language for describing coordination in multi agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems*, 1995.

Deepika Chauhan. *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation.* PhD thesis, ECECS Department, University of Cincinnati, 1997.

Francisco J. Martin, Enric Plaza, and Josep L. Arcos. Interagents: Providing knowledge representation languages with agent-oriented capabilities. 1998. Submitted.

R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. R. Gruber, and R. Neches. The darpa knowledge sharing effort: Progress report. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992.

Juan A. Rodriguez-Aguilar, Francisco J. Martin, Pablo Noriega, Pere Garcia, and Carles Sierra. Competitive scenarios for heterogenous trading agents. In *Second International Conference on Autonomous Agents*, 1998.

[3] `http://www.iiia.csic.es/Projects/fishmarket`