An Extended Set of Interaction Primitives for Multi-agent Systems Development

Mariano Tucat Alejandro J. García

Artificial Intelligence Research and Development Laboratory
Department of Computer Science and Engineering
Universidad Nacional del Sur
Av. Alem 1253, (8000) Bahía Blanca, Argentina
e-mail: {mt,ajg}@cs.uns.edu.ar

The research line reported here involves the improvement of the set of interaction primitives that we have developed [8, 2] for the implementation of multi-agent systems in dynamic and distributed environments. This interaction primitives have been implemented for allowing the creation of several independent multi-agent systems, where agents communicate with others just by knowing the other agents' names. The framework includes primitives for associating the arrival of a message with the automatic execution of a Prolog predicate, thus allowing event-based programming. It also allows the implementation of standard Agent Communication Languages, and provides tools for developing standard Agent Conversation Protocols. We will first describe briefly our framework and then we will propose some improvements for it.

1 The Current Framework

The interaction primitives that we have devised were motivated by the implementation of multi-agent systems for dynamic and distributed environments, where intelligent agents communicate and collaborate. We have used different versions of Prolog for implementing such systems [4]. Although some of the existing Prolog systems provide tools for different subsets of our requirements, none of them provide all the features present in our framework.

For the design and implementation of the proposed primitives, we have remained close to the spirit of Logic Programming: *to provide a specification of the solution and to hide as much of the implementation details as possible.* Thus, these primitives provide a transparent way for programming agent interaction; this can be done, for example, by using the agents' logical names without considering low level elements like the actual location of an agent, IP addresses or machine names.

As we have already mentioned, our first goal was to specify a set of primitives which will provide a framework for implementing agent interaction in dynamic and distributed environments. The resulting primitives allow the implementation of standard Agent Communication Languages like FIPA ACL [1] and KQML [3], and provide tools for developing standard Agent Conversation Protocols. Figure 1 shows the list of the implemented primitives. Observe that some of them were inspired by PVM [6], MPI [5], BinProlog and Jinni [7].

Partially supported by SGCyT Universidad Nacional del Sur, CONICET and Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 Nro 13096)

Primitive:	Type:	Brief description:
connect	setup	Agent initialization and connection to a generic MAS
connect_at	setup	Connection to a particular MAS
disconnect	setup	Deletes the agent from the MAS of which it is a member
my_name	setup	Returns the agent's public name
which_agents	setup	Returns the rest of the participants in the MAS
send	msg.	Sends a message to one or more agents
receive/3	msg.	Waits for a message from another agent
receive/4	msg.	Waits for a message for a given period of time
bind	events	Binds the messages of a specific agent to the call of a predicate
bind_all	events	Binds the messages of any agent to the call of a predicate
unbind	events	Unbind the messages of a specific agent
unbind_all	events	Unbind the messages of any agent
remote_run	exec.	Initiates the execution of a predicate in another connected agent
remote_call	exec.	Consults any other connected agent's knowledge base

Figure 1: Brief description of the set of implemented primitives for agent interaction

Our second goal was to obtain an implementation of this framework that would provide an abstraction layer for the programmer, hiding as much of the low level details as possible (IP addresses, machine names, *etc.*). Thus, our framework design remains close to the Logic Programming paradigm, providing a transparent way for implementing interaction among agents. The primitives in Figure 1 were implemented in a module that can be loaded and used in a free public domain Prolog.

The resulting framework has the following features:

1. The implemented primitives allow the creation of several independent multi-agent systems inside a LAN.

2. An agent that joins a MAS can communicate with the other participants by just knowing their names, regardless of which machine they are actually in.

3. Once an agent runs the initialization predicate (`connect`) it obtains a list of the agents present in the system, and thereafter it may send/receive messages in a very simple manner.

4. The basic communication primitives (`send`/`receive`) allow the exchange of Prolog terms.

5. There are primitives (`bind` or `bind_all`) for associating the arrival of a message with the automatic execution of a Prolog predicate, thus allowing event-based programming.

6. Different associations can be made for different agents.

7. An agent can request the remote execution (`remote_run`) of a particular predicate from another agent.

8. An agent can consult another agent's knowledge base (`remote_call`) and use backtracking in order to obtain multiple answers.

2 New Features and Extensions

The framework presented above has some limitations. In order to solve them and also improve the framework, we are planning to extend it in several ways. In this section we briefly describe these possible improvements.

As mentioned above, event-driven programming can be done by associating the arrival of a message from an specific agent to the automatic execution of a Prolog predicate. Thus an agent may associate the messages from different agents to different predicates. Although this is useful in some applications, there exists another situations in which it would be better to allow an agent to associate the arrival of different types of messages to the automatic execution of different predicates.

As we have already mentioned, interaction is an essential characteristic of Multi-Agent Systems (MAS). This interaction is usually performed by exchanging messages according to some conversation policy. In order to do this, an agent may use standard Agent Communication Languages like the one proposed by FIPA. In this case, allowing the association of different types of messages to the execution of different predicates facilitate the development of agents. Thus, an agent may associate specifics predicates to the arrival of messages from specifics protocols.

Another feature of our framework is that it allows the creation of several independent multi-agent systems. An agent may join any of them and it may also leave a MAS in order to join another one. However it can not be member of more than one multi-agent system simultaneously. We are planning to extend the framework in order to allow the agents to be member of multiple multi-agent systems.

This would be useful in some situations in which there exists different multi-agent systems and some agents wants to be member of more than one of them and also interact with the participants of the multi-agent systems simultaneously. For example, in a soccer game, each team is a different multi-agent system and the agents part of one of this MAS may not interact with the agents of the other MAS. However, there may exists the agent referee that may interact with the agents of both multi-agent systems simultaneously.

Other characteristic of our framework is that the primitives implemented assume that the agents will not be malicious in any sense. Thus, an agent may join any of the existing multi-agent systems. This represents a limitation concerning security aspects, which we are planning to solve allowing the creation of private multi-agent systems. The access to this multi-agent systems should be restricted to agents that have the corresponding key or password. In the previous example, the MAS corresponding to each team should restrict the access to the agents that are part of the team.

Another security aspect not considered yet is that an agent may execute any predicate in any member of the MAS (using `remote_run/3`) and it may also consult the other agent knowledge base (using `remote_call/3`). We are planning to solve this situation by adding primitives to limit the predicates that can be executed remotely. This means that an agent may use this primitives to allow other agents to execute or call remotely an specific predicate. Those predicates that the agent allows to be executed by the participants of the MAS can be seen as services and they will be specified dynamically.

Allowing the remote execution or call of predicates that already exists in another agent is useful in some applications. However, there exists some situations in which an agent has a predicate that it wants to execute in another agent knowledge base. In this case, the agent may send the predicate to the other agent, calls it remotely and obtains its' result. We are considering the possibility to extend the framework in order to have a primitive that allows this kind of remote execution, having also in mind the security aspects.

Following with this idea, it may be useful not only to move a predicate to be run remotely but also to move the whole agent and execute it in another machine, allowing agent mobility. This means that an agent running in an specific machine may pause its execution, move its code and state to another machine and resumes its execution there, in a transparent way. Neither the participants of the MAS should realize that the agent has moved from one machine to another, nor the agent should lose any message or request sent to it.

The framework should determine the best host to move the agent based on some parameters. One of this parameters could be the number of agents per host, allowing a better distribution of agents among hosts in order to achieve maximum efficiency. Another possible parameter may be the messages interchanged between the agent and the rest of the participants of the MAS.

3 Conclusions

The interaction primitives that we have devised were motivated by the implementation of multi-agent systems for dynamic and distributed environments, where intelligent agents communicate and collaborate. The implemented primitives allow the creation of several independent multi-agent systems. Agents may communicate with others just by knowing the other agents' names. The basic communication primitives (`send` and `receive`) allow the exchange of Prolog terms. Event-driven programming can be done associating the arrival of a message with the automatic execution of a Prolog predicate. An agent can request from another agent the remote execution of a particular predicate.

However, this set of interaction primitives has some limitations. In order to solve them and also to improve the framework, we propose here some possible extensions. The present framework does not covered some specific situations, like those when agents may belong to several MAS. Thus, we proposed an extension to allow the agents to be member of multiple multi-agent systems. The framework presented also limits an agent to bind every message of another agent to the execution of only one predicate. Here we propose an extension to allow an agent associate the arrival of different types of messages to the automatic execution of different predicates. In order to consider security aspects, we also proposed improvements to allow the creation of private multi-agent systems and also to allow the agents to limit the predicates that can be executed remotely by the other agents.

References

[1] FIPA. Foundation for intelligent physical agents. http://www.fipa.org.

[2] Alejandro J. García, Mariano Tucat, and Guillermo R. Simari. Interaction Primitives for Implementing Multi-agent Systems in Prolog. 2005.

[3] KQML. Knowledge Query and Manipulation Language. Oficial Web Page: http://www.cs.umbc.edu/kse/kqml.

[4] LIDIA. Artificial Intelligence Research and Development Laboratory. Department of Computer Science and Engineering Universidad Nacional del Sur. Bahía Blanca, Argentina. http://cs.uns.edu.ar/lidia.

[5] MPI. Message Passing Interface. http://www-unix.mcs.anl.gov/mpi.

[6] PVM. Parallel Virtual Machine. http://www.csm.ornl.gov/pvm/pvm_home.html.

[7] Paul Tarau. Jinni 2002: A High Performance Java and .NET based Prolog for Object and Agent Oriented Internet Programming. http://www.cs.unt.edu/∼tarau/.

[8] Mariano Tucat. *"Primitivas de Interacción para el Desarrollo de Sistemas Multi-agente"*. Proyecto Final de Ingeniería en Sistemas de Computación. Computer Science Department, Universidad Nacional del Sur, Bahía Blanca, Argentina, March 2005. http://cs.uns.edu.ar/∼ajg/interaction_primitives/.