

Computación y Sistemas
ISSN: 1405-5546
computacion-y-sistemas@cic.ipn.mx
Instituto Politécnico Nacional
México

Martin, Francisco J.; Plaza, Enric; Rodríguez Aguilar, Juan A.
Composing Multi-Agent Systems with Interagents
Computación y Sistemas, vol. 2, núm. 4, abril-junio, 1999, pp. 183-190
Instituto Politécnico Nacional
Distrito Federal, México

Available in: http://www.redalyc.org/articulo.oa?id=61520405

How to cite
Complete issue
More information about this article
Journal's homepage in redalyc.org

redalyc.org
Scientific Information System
Network of Scientific Journals from Latin America, the Caribbean, Spain and Portugal
Non-profit academic project, developed under the open access initiative

Computación y Sistemas Vol. 2 No. 4 pp.183 - 190
© 1999, CIC - IPN. *ISSN 1405-5546* Impreso en México

Composing Multi-Agent Systems with Interagents

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

Article received on November 15, 1998; accepted on April 19, 1999

Abstract

This paper presents an infrastructure for easing the construction of agent-based systems. Our proposal relies upon two basic elements: conversation protocols, coordination patterns that impose a set of rules on the communicative acts uttered by the agents participating in a conversation (what can be said, to whom, and when), and interagents, autonomous software agents that mediate the interaction between each agent and the agent society. We also introduce JIM, *our java-based implementation of a general-purpose interagent that employs conversation protocols for mediating conversations among agents. Finally, we illustrate the use of interagents by explaining the several roles that they play in FM* (Rodríguez-Aguilar and Martin, 1999), *an agent-mediated electronic auction market.*

Keywords:

Multi-Agent Systems, Interagents, Communication, Coordination.

1 Introduction

There exists a number of problems which involve multiple and (logically and often spatially) distributed sources of knowledge. This kind of problems can best be addressed using an agent-based system —a computational system composed of several interacting agents which cooperate with one another to solve complex tasks beyond the capabilities of an individual agent. The development of such type of multi-agent systems poses the question of how to integrate a set of heterogeneous agents —agents developed by different people for different purposes and in different languages— within a common setting. We have regarded such process as composed of two well-defined, separate stages concerning respectively:

- *the agents' logics*: involving from the agents' inner behavior (knowledge representation, reasoning, learning, etc.) to the agents' social behavior responsible for high-level coordination tasks (the selection, ordering, and communication of the results of the agent activities so that an agent works effectively in a group setting).
- *the agents' interactions* taking place at several levels: *content level*, concerned with the information content communicated among agents; *intentional level*, expressing the intentions of agents' utterances, usually as performatives of an agent communication language (ACL), e.g. KQML, FIPA ACL, etc.; *conversational level*, concerned with the conventions shared between agents when exchanging utterances; *transport level*, concerned with mechanisms for the transport of utterances; and *connection level*, concerned with network protocols (TCP/IP, HTTP, etc.).

Based on this separation of concerns, we have introduced an *infrastructure* that eases the construction of agent-based systems by taking charge of the

cumbersome interaction issues inherent to this type of systems. Such infrastructure relies upon two fundamental elements: *conversation protocols*, coordination patterns that impose a set of rules on the communicative acts uttered by the agents participating in a conversation (what can be said, to whom, and when), and *interagents*, autonomous software agents that mediate the interaction between each agent and the agent society wherein this is situated. In our infrastructure interagents constitute the unique way through which agents interact within a multi-agent scenario (for instance see Figure 1). Thus, as a rule, agents' external behavior is managed by interagents, while their individual logic, knowledge, reasoning, learning, etc. are internal to agents.

We have materialized the conceptualization of interagents through the design and development of JIM: a java-based implementation of a general-purpose interagent capable of managing conversation protocols and capable also of dealing with agent interaction at different levels. JIM has been successfully applied in the development of FM (Rodríguez-Aguilar and Martin, 1999), our current implementation of an agent-mediated electronic auction market.

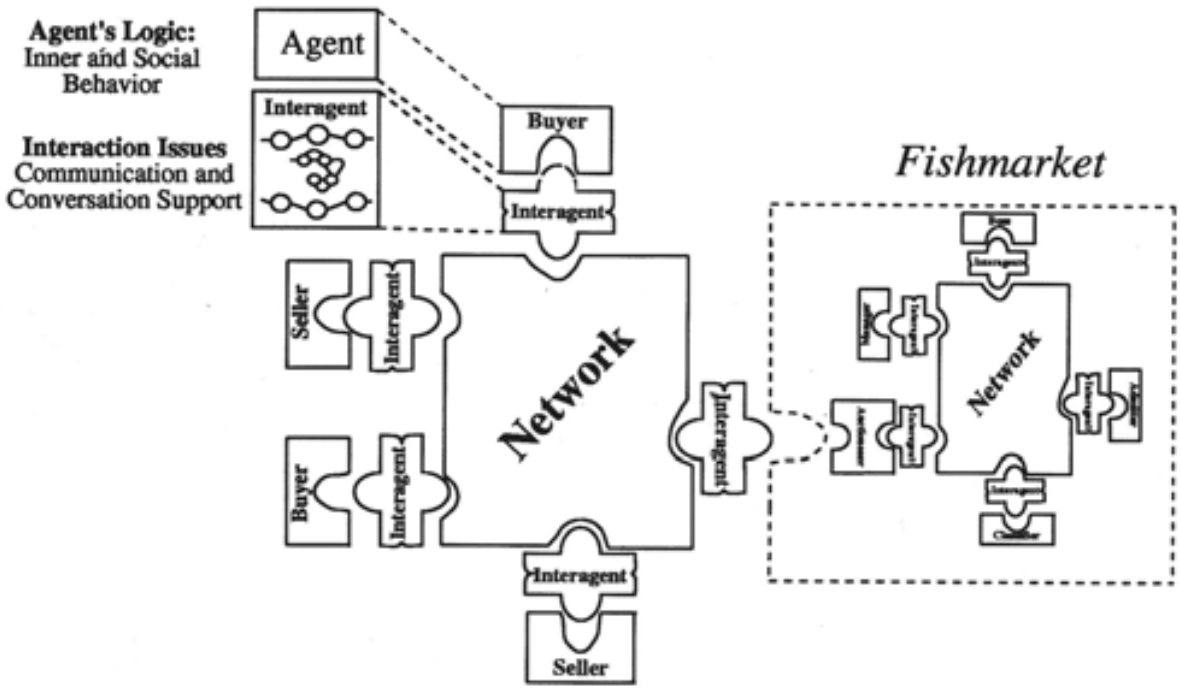


Figure 1: The Fishmarket, an electronic auction house where interactions among agents (trading agents and market intermediaries) takes place through their corresponding interagents.

The remainder of this work is organized as follows. Section 2 overview conversation protocols. Section 3 explains the basics of interagents, introduces JIM, and sketches out the SHIP protocol employed to handle agent interaction at different levels. Section 4 illustrates how interagents are being used in the development of FM. Finally, Section 5 analyzes which features distinguish our approach from related approaches and Section 6 draws some concluding remarks.

2 Conversation Protocols

We view conversations as the means of representing the conventions adopted by agents when interacting through the exchange of utterances —"utterance suggests human speech or some analog to speech, in which the message between sender and addressee conveys information about the sender" (Parunak, 1996). More precisely, such conventions define the *legal* sequence of utterances that can be exchanged among the agents engaged in a conversation: what can be said, to whom and when. Therefore, *conversation protocols* are coordination patterns that constrain the sequencing of utterances during a conversation.

A conversation protocol (CP) defines a class of legal sequences of utterances that can be exchanged between two agents holding a conversation. We model and implement a CP as a special type of Pushdown Transducer (PDT), which can be seen in turn as a

combination of a Finite-State Transducer (FST) and a Pushdown Automaton (PDA): an FST is simply a Finite State Automaton (FSA) that deals with two tapes. To specify an FST, it suffices to augment the FSA notation so that labels on arcs can denote pairs of symbols; whereas a PDA is composed of an input stream and a control mechanism —like an FSA— along with a stack on which data can be stored for later recall.

Therefore, a PDT is essentially a pushdown automaton that deals with two tapes. A PDA can be associated to a PDT by considering the pairs of symbols on the arcs as symbols of a PDA. The choice of PDTs as the mechanism for modeling CPs is motivated by several reasons. First, analogously to other finite-state devices a few fundamental theoretical basis make PDTs very flexible, powerful and efficient. They have been largely used in a variety of domains such as pattern matching, speech recognition, cryptographic techniques, data compression techniques, operating system verification, etc. They offer a straightforward mapping from specification to implementation. PDTs, unlike other finite state devices, allow us to store, and subsequently retrieve, the contextual information of ongoing conversations.

In (Martin *et al.*, 1999) we explained CPs in depth. We also introduced a conceptual model of CPs, the formalism underpinning such model in order to provide the foundations for a further analysis concerning the properties that CPs must exhibit and, finally, the way of negotiating, instantiating, and employing CPs.

3 Interagents

Interagents mediate the interaction between an agent and the agent society wherein this is situated. The main task of an interagent is the management of conversation protocols. Here we differentiate two roles for the agents interacting with an interagent: *customer*, played by the agent exploiting and benefiting from the services offered by the interagent; and *owner*, played by the agent endowed with the capability of dynamically establishing the policies that determine the interagent's behavior. Needless to say that an agent can possibly play both roles at the same time. Moreover, several owners can even share their property (*collective ownership*), whereas several customers can make use of the same interagent (*collective leasing*).

In what follows we provide an account of the functionality of interagents —mostly from the point of view of customers— to illustrate how they undertake conversation management. An interagent is responsible for posting the utterances of its customer to the corresponding addressee and for collecting the utterances that other agents address to its customer. This *utterance management* abstracts customers from the details concerning the agent communication language and the network protocol. Each interagent owns a collection of relevant conversation protocols (CP) used for managing its customer conversations. When its customer intends to start a new conversation with another agent the interagent instantiates the corresponding conversation protocol. Once the conversation starts, the interagent becomes responsible for ensuring that the exchange of utterances conforms to the CP specification.

When starting a new conversation the interagent performs a *CP negotiation* process with the interagent of the addressee agent. The goal of CP negotiation is to reach an agreement with respect to the specific conversation protocol to be used. Moreover, before starting a conversation, the interagent performs a *CP verification* process. This process checks whether the CP to be used verifies the necessary conditions (liveliness, termination, deadlock and race condition free) for guaranteeing the correct evolution of an interaction. Finally, an interagent allows its customer to hold several conversations at the same time. This capability for *multiple conversations* is important because, although in the paper we consider only conversations with two participants (dialogues), conversations with any number of participants are built as a collection of simultaneous CP instances. In other words, the agent views a conversation as involving n participants · while its interagent views such conversation as a collection of simultaneous dialogues represented as multiple CP instances.

3.1 JIM

JIM is a general-purpose interagent enabled with the capability of managing CPs wich has proven its usefulness in several applications (see Section 4). Before the presentation of JIM we will introduce the SHIP protocol because it has profoundly shaped its architecture.

The extenSible and Hierarchical Interaction Protocol (SHIP) offers a layered approach to support all levels of agent interaction (one layer for each level introduced in Section 1). Each layer comprises a set of protocols with similar functionality, and provides an abstract interface to higher layers that hides the details related to the particular protocol in use. For instance, two agents can exchange utterances without worrying about the underlying communication mechanism —message passing, tuple-space or remote procedure call. In addition to this, the hierarchical structure of SHIP allows agents to choose their interaction level. This implies the use of the corresponding layer and the lower layers, but not higher layers, for instance if an agent decides to use the transport layer it will use the lower levels (the connection layer in this case) but not the higher layers of

SHIP. Lastly, SHIP is extensible, in the sense that it permits the incorporation (plug-in) of new protocols into each layer.

SHIP is a request/response protocol which distinguishes two roles: client and server. An interagent may play both roles at the same time, e.g. it can act as a server for its customer, and as a client for another interagent. In a typical message flow a client sends a request to the server, and this sends a response back to the client. All messages exchanged between a client and a server are encapsulated as either requests or responses of the SHIP protocol. Both requests and responses are represented as XML messages that nest the information corresponding to each level of interaction. On the sender side, messages to be sent are wrapped at each layer and passed down till getting to the communication layer responsible for the sending. Upon reception, on the receiver side, messages are unwrapped at each layer and passed up to higher layers till the message arrives at the layer chosen by the agent for interacting.

The architecture of JIM (Figure 2 shows a sketch of it) presents up to four different façades: *owner*, *customer*, *society* and *web* façades. Each facade represents a different entry for each type of agent requesting the services provided by JIM. The services provided through each façade differ. For instance, only the owner façade allows an agent to upgrade the CP repository. The web façade simply provides access to World-Wide Web resources. For instance, this façade allows a human agent to interact with the JIM's customer through a standard WWW browser. Each façade is built on top of a different instance of the SHIP protocol. In JIM a chain of responsibility is established through the different protocol layers that conform the SHIP instance of each façade. That is to say, each layer is responsible for managing some of the services provided by JIM. We explain the rest of the architecture of JIM in terms of its components.

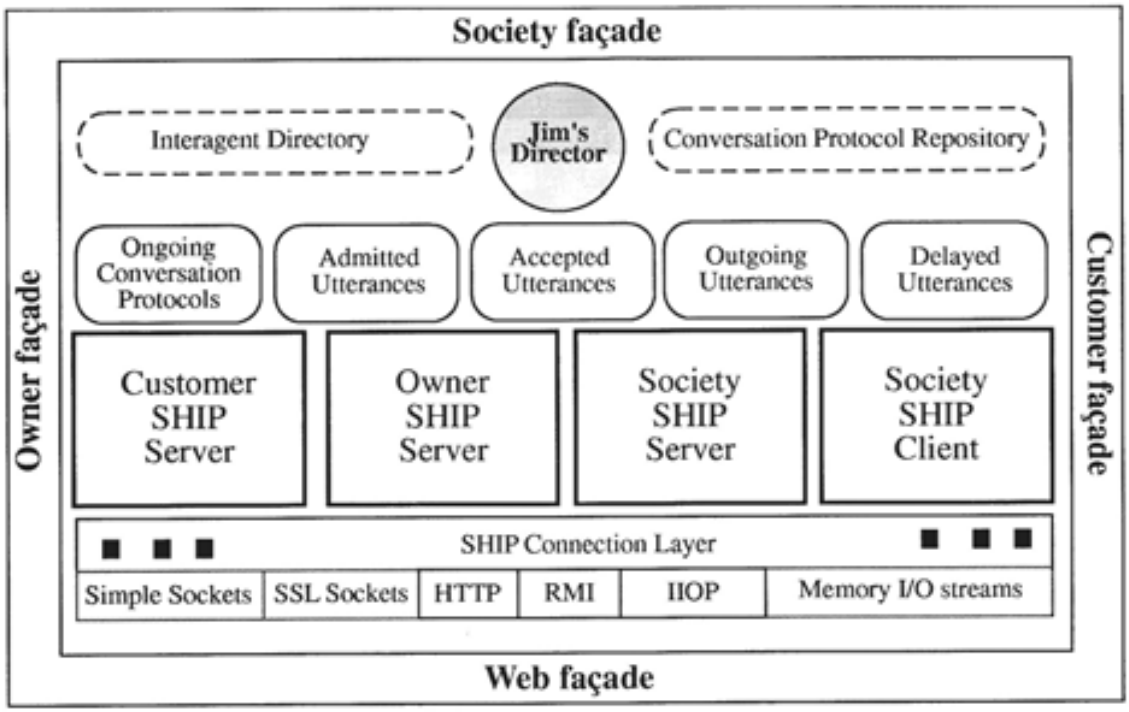


Figure 2: An Overview of JIM's Architecture.

The Interagent Directory component offers interagent naming services (white pages) that translate from unique identifiers of interagents into their logical addresses and vice versa. Each interagent can communicate with a subset of interagents of the agent society, named its *acquaintances*. A reflexive and symmetric, but not transitive, relation acquaintance-of can be established among interagents. An interagent's acquaintanceship is a set of unique identifiers and can vary dynamically, allowing the collection of agents to grow dynamically. The Director component directs the behavior of the rest of components and can receive specific directives from

both the interagent's owner and customer. Each CP defines a set of constraints that the director takes into account to sort the utterances stored in the different buffers (see below). The CP repository component stores all CP classes that can be instantiated by an interagent. Such repository can be updated in either *statically* or *dynamically*, e.g. in FM (see Section 4 only the market intermediaries working for the auction house can define and store at run-time new CP classes into the CP repository of each interagent. Notice that the conversation protocol repository can either be owned by only one interagent or shared among several interagents. The Ongoing conversation protocols component stores all running CPs. A CP instance is kept for each conversation in which the interagent's customer is involved. Once a conversation is over the CP is thrown away (or alternatively kept for tracing purposes). The Buffer for admitted utterances component is responsible for storing all utterances coming from other interagents in the agent society. An utterance is stored here until either it is accepted or it expires due to a time-out condition. The Buffer for accepted utterances component stores utterances that have already been accepted by a CP but which have not been required by the customer yet. Once one of these utterances is required by the customer it is packaged up as a SHIP response and forwarded to it. The Buffer for outgoing utterances component stores utterances that have been forwarded from a customer agent, and received by its interagent but not yet accepted. Notice that the input list of each CP instance can be seen as a composition of the buffers for both incoming and ongoing utterances. The Buffer for delayed utterances component stores the utterances that have been sent by the customer specifying a lapse of time before they are actually transmitted. When such lapse of time expires, they are automatically transferred to the buffer for outgoing utterances.

4 The Agent-based Electronic Auction Market FM

In this section, we introduce FM an agent-based system application for e-commerce that will serve to illustrate both the practical use and functionality of interagents (Rodríguez-Aguilar and Martin, 1999). In fact, most of the features of interagents have emerged as a generalization of the more ad-hoc notion of remote control introduced at the first stages of the development of FM (Rodríguez-Aguilar *et al.*, 1997). FM is an electronic auction house, based on the traditional fish market auctions. Moreover, FM has been extended to produce the multi-agent test-bed for trading agents in electronic auctions (Rodríguez-Aguilar et al., 1998). The current version of FM is now available and can be downloaded from the FishMarket project web page[1].

The actual fish market is an *institution* that establishes and enforces explicit conventions. In our approach, we have described and modeled it as an electronic institution where several scenes run simultaneously, at different places, but with some causal continuity (Rodríguez-Aguilar *et al.*,1997). The principal scene is the auction itself, in which buyers bid for boxes of fish that are presented by an auctioneer who calls prices in descending order —the *downward bidding protocol*. However, before those boxes of fish may be sold, fishermen have to deliver the fish to the fish market, at the *sellers' registration scene*, and buyers need to register for the market, at the *buyers' registration scene*. Likewise, once boxes of fish are sold, the buyers take it away by passing through a *buyers' settlements scene*, while sellers collect their payments at the *sellers' settlements scene* once their lot has been sold. Each scene is supervised by one of the market intermediaries (auctioneer, buyer admitter, buyer manager, seller admitter, seller manager, and boss) which work for and represent the institution.

In a highly mimetic way, the FM electronic market also involves the concurrency of several scenes governed by the market intermediaries identified in the fish market. Therefore, seller agents register their goods with a seller admitter agent, and can get their earnings (from a seller manager agent) once the auctioneer agent has sold these goods in the auction room. Buyer agents, on the other hand, register with a buyer admitter agent, and bid for goods which they pay through a credit line that is set up and updated by a buyer manager agent. Figure 3 shows the conceptual model of FM as a multi-agent scenario.

Similarly to the actual fish market, FM can be regarded as an *electronic institution*. Thus, buyer and seller agents can trade goods as long as they comply with the *Fishmarket institucional* conventions. Those conventions that affect buyers and sellers have been coded into CPs handled by interagents that establish what utterances can be said by whom and when. We shall differentiate two types of interagents in FM: *trading interagents*, owned by the institution but used by trading agents, and *institutional interagents*, both owned and used by those agents functioning as market intermediaries. Therefore, all interagents have the institutions as *owner*, but their *customers* can be either trading agents or the institution itself.

Trading interagents constitute the sole and exclusive means through which trading agents interact with the market intermediaries representing the institution. Within each scene, a trading interagent must employ a different CP to allow its customer to talk to the market

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

intermediary in charge of it. Trading interagents are responsible for enforcing the protocols that guarantee that every trading agent behaves according to the rules of the market. In general, we shall refer to the conversations supported by trading interagents as *intra-scene conversations* because they are dialogues held within each scene between the market intermediary responsible of the scene and each trading agent situated in the scene.

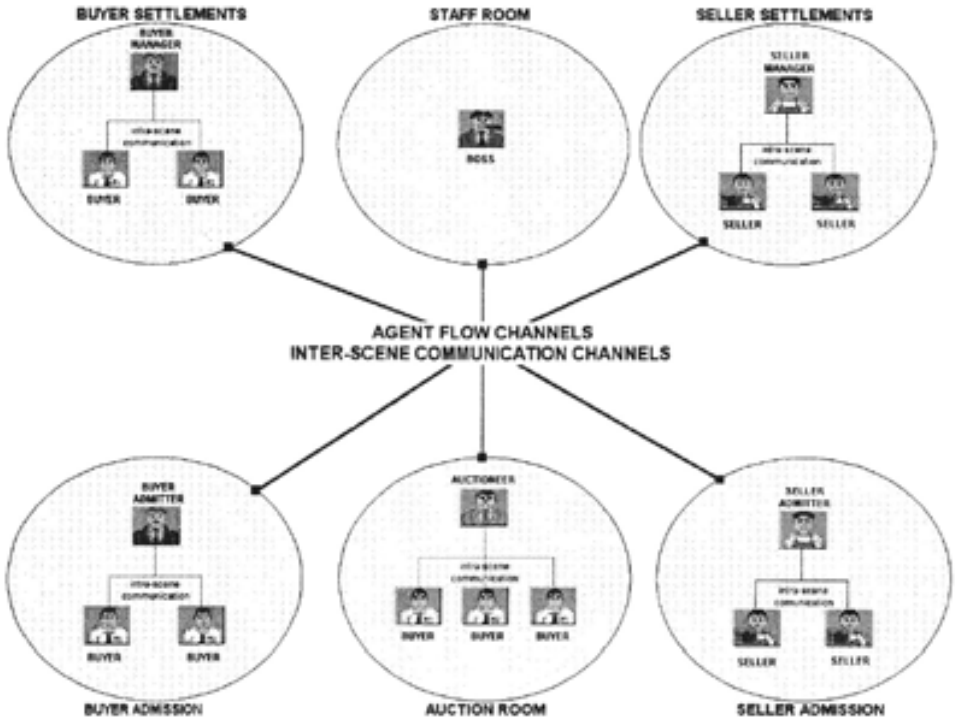


Figure 3: FM Multi-scene Structure.

As to market intermediaries, they must hold several conversations at the same time with the agents in the scene that they govern. For this purpose, they exploit the capability for supporting multiple conversations of interagents, as explained in Section 3, by building collections of simultaneous CP instances (one per trading agent). Thus, for example, the auctioneer's interagent maintains an instance of the CP/DBP for each buyer agent in the auction room. Moreover, institutional interagents are not only used to support conversations with trading agents, but also to allow those agents working as market intermediaries to coordinate their activities. We will refer to the conversations held between market intermediaries as *inter-scene conversations*.

Both types of interagents have been endowed with further capabilities. On the one hand, they are in charge of conveying monitoring information to the auditing agent (also called monitoring agent), so that market sessions can be monitored, and analyzed step-by-step. On the other hand, interagents provide support for agent failure handling. For example, when a trading agent either goes down or simply fails to consume the utterances conveyed by its interagent (say that the trading agent lapses into an extremely demanding deliberative process for elaborating its strategies), this interagent pro-actively unplugs its customer from the market and leaves the market on its behalf. Finally, by applying the results deriving from our analysis in (Martin *et al.*, 1999), the interagents in FM can dynamically (at run-time) reconfigure retrieved CPs in order to guarantee the verification of liveliness, termination, and deadlock and race condition free.

Summarizing, the successful incorporation of interagents into FM has proven their usefulness by coping with several tasks: *i)* to handle the interplay between trading agents and the market institution (intra-scene conversations); *ii)* to handle the coordination between market intermediaries' tasks (inter-scene conversations); *iii)* to provide support for the monitoring of market sessions; *iv)* to handle agents' failures; *v)* to reconfigure CPs so as to ensure protocol compatibility.

5 Related Work

So far, much effort in agent research concerning agent interaction has focused on the semantic and pragmatic foundations of different agent communication languages (ACLs) based on speech act theory (Searle, 1969; Cohen and Levesque, 1995). However this research is currently broadening from the specification of individual utterances to include the characterization of goal-directed conversations for which agents will use ACLs. Thus new works in speech act research, exemplified by efforts such as KaoS (Bradshaw, 1996), Dooley Graphs (Parunak, 1996), COOL (Barbuceanu and Fox, 1995) and MAGMA (Demazeau, 1995), attempt at representing and reasoning about the relationships within and among conversations, or groups of utterances. A number of formalisms have been proposed for the modeling of conversations: FSMs (Chauhan, 1997), Dooley graphs, Petri Nets, etc. Our approach proposes a new model based on PDTs that allows to store the context of ongoing conversations, and, in contrast with other approaches, that provides a mapping from specification to implementation.

Although there is a large number of software tools for developing agents, not many of them happen to provide support for the specification of conversation protocols. Agent Talk, COOL (Barbuceanu and Fox, 1995), JAFMAS (Chauhan, 1997), Agentis (d'Inverno *et al.*, 1998), Jackal (Cost *et al.*, 1998) and InfoSleuth (Nodine *et al.*, 1998), do offer conversation constructs. AFMAS, for instance, provides a generic methodology for developing speech act based multi-agent systems using coordination constructs similar to COOL. In addition to this, as far as our knowledge goes, none of them offers dynamically and incrementally specifiable conversation protocols except for InfoSleuth. We attempt to make headway in this matter with respect to other agent building tools by introducing interagents, autonomous software agents that permit both the dynamic and incremental definition of conversation protocols by both the agent engineer and the owner.

We have chosen the conceptualization of interagents as autonomous software agents instead of an agent's built-in conversation layer as proposed in other agent architectures because of the need to separate the agents' logics from the agents' interactions —such separation has proven to be valuable in the development of a particular type of agent-based systems, namely electronic institutions such as FM. Interagents —like KQML facilitators— are inspired by the *efficient secretary* metaphor already introduced in the Actors model of concurrent computation.Nonetheless, *interagents* —unlike KQML facilitators— offer the conversational level required by agents to cooperate in non-trivial ways.

6 Final Remarks

Based on CPs and interagents we have proposed an *infrastructure* for easing the development of agent-based systems that takes charge of the complex, time-consuming interaction issues inherent to the construction of this type of systems. In this way, the overhead related to the management of the interaction tasks needed by an agent is shifted to its interagent. Interagents employ CPs for mediating conversations among agents, and so the management of CPs is identified as their main task. Two major benefits are achieved by deploying our infrastructure from the point of view of the agent developer: on the one hand, their agents can reason about communication at higher levels of abstraction, and on the other hand they are released from dealing with interaction issues, and so they can concentrate on the design of the agents' logics. We have also introduced JIM, a general-purpose interagent. JIM is currently being used to coordinate the activities of the market intermediaries composing the FM system and the interaction between the market as a whole and the participating buyers and sellers (see Figure 1). Additionally, JIM is being successfully employed by other ongoing research projects: the SMASH project[2], that addresses the construction of prototype multi-agent systems with learning capabilities that cooperate in the solution of complex problems in hospital environments; and in the multi-agent learning framework Plural (Martin *et al.*, 1998) which tackles the problem of sharing knowledge and experience among cognitive agents that co-operate within a distributed case-based reasoning framework.

Acknowledgments

This work has been supported by the Spanish CICYT project SMASH, TIC96-1038-C04001; the COMRIS project, ESPRIT LTR 25500; Juan Antonio Rodríguez and Francisco J. Martin enjoy DGR-CIRIT doctoral scholarships FI-PG/96-8490 and FI-PG/96-8472 respectively.

References

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

Bradshaw, J.M., "Kaos: An open agent architecture supporting reuse, interoperability, and extensibility", in *Tenth Knowledge Acquisition for Knowledge Based Systems*, 1996.

Chauhan, D., *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation,*

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

PhD thesis, ECECS Department, University of Cincinnati, 1997.

Cohen, P.R. and **Levesque, H.J.**, "Communicative actions for artificial agents", in *Proceedings of the First International Conference on Multi-Agent Systems* (ICMAS-95), Menlo Park, CA. AAAI Press., 1995, pp. 65-72.

Cost, R.S., Finin, T., Labrou, Y., Luan, X., Peng, Y., and **Soboroff, I.**, "Jackal: a java-based tool for agent development", in *AAAI-98 Workshop on Software Tools for Developing Agents*, 1998.

Demazeau, Y., "From interactions to collective behaviour in agent-based systems", in *European Conference on Cognitive Sciences*, 1995.

d'Inverno, M., Kinny, D., and **Luck, M.**, "Interaction protocols in agentis", in *Third International Conference on Multi-Agent Systems*, 1998.

Martin, F.J., Plaza, E., and **Arcos, J.L.**, "Knowledge and experience reuse through communication among competent (peer) agents", to appear in *International Journal of Software Engineering and Knowledge Engineering*, 1998.

Martin, F.J., Plaza, E., and **Rodriguez, J.A.**, "An infrastructure for agent-based systems: an interagent approach", to appear in *International Journal of Intelligent Systems*, 1999.

Nodine, M., Perry, B., and **Unruh, A.**, "Experience with the infosleuth agent architecture", in *AAAI-98 Workshop on Software Tools for Developing Agents*,1998.

Parunak, H. V. D., "Visualizing agent conversations: Using enhanced dooley graph for agent design and analysis", in *Proceedings of the Second International Conference on Multi-Agent Systems*, 1996.

Rodríguez-Aguilar, J.A. and **Martin, F.J.**, "Auction-based electronic markets for human and software traders", *Simposio Español de Informática Distribuida* (SEID'99), 1999.

Rodríguez-Aguilar, J., Martin, F.J., Noriega, P., Garcia, P., and **Sierra, C.**, "Competitive scenarios for heterogenous trading agents", in *Second International Conference on Autonomous Agents*, 1998.

Rodríguez-Aguilar, J.A., Noriega, P., Sierra, C., and **Padget, J.**, "Fm96.5 a java-based electronic auction house", in *Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology* (PAAM'97), 1997.

Searle, J., *Speech Acts*, Cambridge University Press, 1969.



Francisco J. Martin *graduated in 1996 in Computer Science at the technical University of Valencia (Spain). He is finishing his PhD at the at the IIIA (Artificial Intelligence Research Institute) of the CSIC (Spanish Scientific Research Council). His research interests lie in the broad areas of Agent-Based Systems, Machine Learning, Component-based Development, Operating Systems, Distributed Processing and Electronic Commerce. He has recently founded iSOCO-Intelligent Software Components, S.A. a spin-off of the IIIA-CSIC that intends to be the nexus between research and development with the purpose of constructing innovative solutions for e-commerce employing artificial intelligence techniques. He can be reached at martin@isoco.com. His homepage is at http://www.iiia.csic.es/~martin*



Enric Plaza i Cervera *is Research Scientist at the IIIA (Artificial Intelligence Research Institute) of the CSIC (Spanish Scientific Research Council). He received his Ph.D. in Computer Science in 1987 at the Universitat Politècnica de Catalunya (UPC) and has worked since on knowledge acquisition and representation, machine learning and case-based reasoning. His main current interest is on multiagent systems with learning capabilities. He has authored a number of journal and conference papers and led some European research projects. He also has organized two international conferences. His homepage is at http://www.iiia.csic.es/People/enric.html*



Juan A. Rodríguez-Aguilar *received the B.Sc. degree in Computer Science and the M.Sc. degree in Artificial Intelligence from the Autonomous University of Barcelona in 1993 and 1998 respectively. He is currently pursuing a Ph.D. in Artificial Intelligence at the Artificial Intelligence Research Institute (IIIA) of the Spanish Scientific Research Council (CSIC). His current research interests lie in the broad areas of agent-oriented software engineering, coordination mechanisms in agent-based systems, and electronic commerce.*

