Coordinating Mobile Agents by means of Communicators

Antonella Di Stefano, Corrado Santoro
University of Catania - Engineering Faculty - Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Viale A. Doria, 6 - 95125 - Catania (ITALY) - Tel. +39 095 7382364/5 - Fax. +39 095 7382397
{adistefa, csanto}@diit.unict.it

Abstract

This paper proposes a coordination model, for both static and mobile agents, based on abstract structures called Communicators*, entities which handle agent dialogue performed through ACL speech act exchanging. Such structures are designed based on a need to model agent dialogue in a human-like style, offering a set of coordination primitives of general validity, able to provide both a direct and indirect interaction model. Since a Communicator handles messages exchanged within a well-defined multi-agent application, it is fully programmable, i.e. it is possible to specify* what *messages can be exchanged and* how *these message have to be handled. In detail, a Communicator performs a* syntactic and semantic routing*, allowing the exchange and forwarding of a message according to the programmed coordination laws.*

1 Introduction

In multi-agent applications, a key characteristic of software agents is the social behaviour [14, 5]. Agents exchange information in order to achieve the global goal of the application to be performed. To model agent interactions, research proposes two approaches: (1) the adoption of *agent communication languages (ACLs)* [10, 7, 8], which are based on agent-to-agent *direct interactions*, and (2) the use of *coordination infrastructures* [3] which offer a support to *indirect interactions* based on shared spaces (e.g. blackboards, tuple spaces, etc.) through which agents interact by placing and retrieving information. Each proposal presents several advantages, but, in the authors' opinion, none of these provide a *complete* framework able to meet all the requirements of a generic multi-agent application. Since agents are software entities exhibiting a sort of human-like behaviour [9], any agent needs to handle well-formed speech acts and, in the majority of cases, to forward the speech act directly to another agent. In other cases, agents need to share information, also using a proper speech act[1], but rather adopting an indirect interaction model able to handle knowledge spreading [6]. In order to provide a coordination infrastructure for agent interaction as complete as possible, we should combine in a single coordination framework both direct and indirect interaction, with the aim to offer a human-like model of agent interaction based on speech act exchange. We must not forget that an agent dialogue occurs in the context of a well-defined multi-agent application which has also a well-defined set of interaction protocols designed during application engineering [13]. These protocols establish the rules of each agent dialogue and can be used to derive the *coordination laws* for the infrastructure which handles agent interaction for that application. To this aim, this paper proposes a coordination model, for both static and mobile agents [4], based on abstract structures called *Communicators*, which represent entities to handle agent dialogue performed through ACL speech act exchange. A Communicator is fully programmable, i.e. it is possible to specify *what* messages can be exchanged and *how* these messages have to be handled. Since each message exchanged passes through a Communicator, the latter performs a *syntactic and semantic routing*, allowing the exchange and forwarding of a message according to the programmed coordination laws. These laws are established by providing *interaction classes*, i.e. a specification of the allowed interaction in terms of the *roles* of the agents involved, *speech act types* and *handling modes*. Communicators can also be composed to permit a flexible specification from coarse to fine-grained.

2 Abstract Model of Agent Interaction

In designing a coordination framework of general validity, we have to analyze the main characteristics of an agent interaction and provide an abstract model of it. In the rest of this paper, we will use the term *agent interaction* to refer to the exchange of a *single message* between two (one-to-one) or more (one-to-many) agents.

[1] e.g. a `tell` or `inform` message without specifying the receiver agent.

According to the methodology described in [13], the first step in designing a multi-agent application is the definition of the *role model* and the *interaction model*, from which interaction protocols are derived and the behaviour of agents in the context of the defined protocols. Given this, we can derive the first property of an agent interaction:

> **Def. 1.** *Any agent interaction between two or more agents participating in the same multi-agent application can take place if (and only if) the role and interaction models of that application allow it.*

More specifically, we can say that a multi-agent application A defines a set of interaction protocols P, say $A = \{P\}$, and each protocol defines a set of allowed interactions I, say $P = \{I\}$; thus we can formalize Def. 1 with the following first-order-logic expression:

$$I' \, can \, occur \, in \, A \Leftrightarrow A = \{P\} \wedge \exists P' \in \{P\} \wedge I' \in P' \tag{1}$$

Now, in practise, let us model an agent interaction by starting from a standard ACL message. This is our starting point as such a message is a good model of the information exchanged in a human interaction. Let us model an ACL message with the tuple:

$$m :=< \sigma_m, \rho_m, n_m, \Omega_m, \lambda_m, \mu_m > \tag{2}$$

where σ_m and ρ_m identify respectively the sender and receiver agent, n_m is the name of the speech act, Ω_m is the ontology, λ_m the content language and μ_m the message content[2]. Such a message model is not complete for representing human-like behaviour. A human interaction can be not only one-to-one, but also one-to-many, and the receivers can be reached by the message in an *explicit* or *implicit* way; for example, in a room where there are some people, if a single person talks *explicitly* to his/her friends, the conversation could be (implicitly) listened to by the other people. This means that we have to consider the agents to which the message is explicitly addressed and the agents which are allowed to "listen to" a message exchanged, even if it is not directed to them. These issues can be modeled by introducing the *receiver set* and the *scope*. They define what we call *real* and *virtual receivers*. Real receivers are the agents to which the message is explicitly destined, and virtual receivers are all the agents which can share the messages. According to this model, we characterize agent message and interaction with the tuples:

$$\begin{aligned} m' &:=< \sigma_m, \{\rho_m\}, n_m, \Omega_m, \lambda_m, \mu_m, \xi_m > \\ \widehat{I} &:=< m', \{v_m\} > \end{aligned} \tag{3}$$

[2]Additional parameters (such as, for a KQML message, `reply-with`, `in-reply-to`, etc.) do not need to be specified since they are not relevant in our model.

With respect to 2, these formulas introduce the possibility of specifying a set of real receivers, $\{\rho_m\}$, instead of a single one. *Virtual receivers* $\{v_m\}$ cannot be under the control of the sender agent: the latter (like a human character) can specify the message to send, who it is, the agents it is speaking to, etc., but the *scope* of the interaction is determined by the application context in which the interaction occurs. For this reason we modelled interactions with $\widehat{I}$ by specifying the association between a message m' with the relevant virtual receivers. This association depends on the applicaton requirements and thus by the *semantics* of that interaction in the context of the given application. In order to allow a flexible specification of such associations, we group all interactions with a same behaviour in *interaction classes* by generalizing each field in $\widehat{I}$. While n_m, Ω_m, λ_m are still generalized, the same cannot be said for agent names and message content: names become *roles*, according to the interaction model of [13], and the content is generalized by using a *message template*. The result is an interaction class $\overline{I}$ represented by the tuple $< \{\sigma\}, \{\rho\}, \{v\}, \{N\}, \{\Omega\}, \{\lambda\}, K >$ where $\{\sigma\}$ are the roles of the sender agent, $\{\rho\}$ the roles of real receivers, $\{v\}$ the roles of virtual receivers, $\{N\}$ the performative names which can be exchanged, $\{\Omega\}$ the allowed ontologies, $\{\lambda\}$ the allowed content languages and K a predicate for the content field (e.g. a regular expression) representing the message template. Given this, we can assert that a message m', modeled as 3, *belongs* to a class $\overline{I}$ if all the fields of the message match with the relevant fields defining the interaction:

$$\begin{aligned} belongs(m', \overline{I}) \quad \Leftarrow \quad & ValidRole(\sigma_m, \{\sigma\}) \wedge \\ & \wedge SValidRole(\{\rho_m\}, \{\rho\}) \wedge \\ & \wedge n_m \in \{N\} \wedge \Omega_m \in \{\Omega\} \wedge \\ & \wedge \lambda_m \in \{\lambda\} \wedge K(\mu_m) \end{aligned} \tag{4}$$

$$ValidRole(r, P) \Leftarrow \exists r' \in P \wedge Role(r) = r' \tag{5}$$

$$SValidRole(R, P) \Leftarrow \forall r \in R \wedge Role(r) \in P \tag{6}$$

Here, the function $Role(\cdot)$ returns the role of the given agent, the predicate $ValidRole(r, P)$ returns true if the role of agent r is defined in set P, and the predicate $SValidRole(R, P)$ is true if the role of each agent in set R is defined in the set P. Now, by combining the interaction class definition with the concepts expressed in Def. 1 and relation 1, we can model an interaction protocol as a *set of interactions* with a well-defined behaviour between agent roles, that is *interaction classes* $\overline{I}$, say $P = \{\overline{I}\}$. Relation 1 thus becomes:

$$m' \, can \, occur \, in \, A \quad \Leftarrow \quad A = \{P\} \wedge \exists P' \in \{P\} \wedge \\ \wedge \exists \overline{I} \in P' \wedge belongs(m', \overline{I}) \tag{7}$$

Virtual receiver set $\{v_m\}$ refers explicitly to message delivery, instead of the simple message exchange occurrence. Thus, to give a stronger formulation of our model, we introduce the predicates *"can be sent"* and *"can be received"* with the following semantics:

$$m' \, can \, be \, sent \, by \, \alpha \, in \, A \quad \Leftarrow \quad m' \, can \, occur \, in \, A \\ \wedge ValidRole(\alpha, \{\sigma\}) \tag{8}$$

$$m' \, can \, be \, recvd \, by \, \alpha \, in \, A \quad \Leftarrow \quad m' \, can \, occur \, in \, A \\ \wedge ((ValidRole(\alpha, \{\rho\}) \wedge \\ \wedge \alpha \in \{\rho_m\}) \vee \\ ValidRole(\alpha, \{v\})) \tag{9}$$

3 Communicators

To provide a coordination infrastructure according to the interaction model of Section 2, we introduce coordination entities called *Communicators*. Let us start with *Simple Communicators*, *SK*, which are entities represented by the triple $SK :=< \overline{I}, \{m\}, \{\alpha\} >$: $\overline{I}$ is an interaction class as defined in Section 2, $\{m\}$ is a set of ACL messages (as 2) temporarily stored in *SK* and $\{\alpha\}$ is the set of agents which have joined to *SK*. A set of primitives provides the operations which can be made on *SK*, allowing an agent to join or leave the *SK* and to read or write a message. Agent communication is performed by exchanging messages through *SK*, the aim of which is to check that the message belongs to the interaction class $\overline{I}$ and then to act as a *semantic router* by forwarding the message to the agent(s), according to the interaction class definition. The basic primitives of *SK* an agent can use are $join_{SK}$, $leave_{SK}$, put_{SK} and get_{SK}. The first one has to be issued by an agent before performing any communicative action (sending or retrieving a message); $join_{SK}$ adds an agent a to the joined agent set $\{\alpha\}$ only if the role of a is defined (as sender or receiver) in the interaction class $\overline{I}$. Subsequently, an agent can send or retrieve a message by using respectively the put_{SK} and get_{SK} primitives: the former inserts the message in the message set $\{m\}$ only if the agent belongs to the joined set $\{\alpha\}$ and the message belongs to $\overline{I}$. get_{SK}, on the other hand, returns a message m' from $\{m\}$, only if the agent a belongs to the joined set $\{\alpha\}$, the role of a is defined as real receiver (the message is *extracted* from $\{m\}$) or as a virtual receiver (the message is *read but not deleted* from $\{m\}$) in $\overline{I}$. To add flexibility to our model, we also introduced the *predicative-get*, p_get_{SK}, which behaves like get_{SK} but returns a message meeting a given predicate. Finally, an agent can leave the Communicator by issuing the $leave_{SK}$ primitive. The behaviour of these primitives is summarized in Figure 1.

A Simple Communicator is able to model the behaviour of an interaction class as defined by 4. The complete interaction framework expressed by 7, 8 and 9 can be modeled by grouping together several Simple Communicators, each one representing a different interaction class defined by the protocols of the application A. To this aim, we introduce *Composite Communicators*, *CK* (or simply *Communicators*) as entities composed of a set of *SK*, $CK := \{SK_1, SK_2, ..., SK_n\}$. A CK provides the same primitives of an SK, but they behave a little bit differently; in detail, $join_{CK}$ joins the agent to each SK_i the interaction class of which defines the role of the agent as sender or receiver; put_{CK} inserts the message in each SK_i which makes true the predicate of put_{SK_i} (see Figure 1); get_{CK} (p_get_{CK}) retrieves a message from an SK_i which makes true the predicate of get_{SK_i} ($p\_get_{SK_i}$); finally, $leave_{CK}$ removes the agents from each joined SK_i.

A so-formed (Composite) Communicator is able to model relation 7, 8 and 9, and thus to provide the coordination infrastructure for a given application defining the interaction classes implemented in the CK. This means that, once a multi-agent application is designed and the interaction classes are defined, we can build the Communicator able to support the desired message exchange in the context of that application. Since the created Communicator represents the entity entailed to handle coordination for a given application, each agent, before participating in the application, has to join the relevant Communicator; subsequently, it can communicate with the other (joined) agents by sending and receiving messages through that Communicator.

The presented model is able to realize both *direct* and *indirect* communication. The former is obtained by specifying explicitly the addressed agent(s); the latter model is obtained by specifying *all* as real or virtual receivers: according to the composition of the sent message, it will be placed in the relevant *SK* and can be read subsequently by any agent belonging to the real or virtual receiver set defined in that *SK*.

3.1 Example: an Electronic Auction

Let us present an example of an electronic auction application for which we will derive the interaction framework modeled according to the principles introduced in the previous section. In this kind of application, we consider three roles: the *auctioneer (A)*, the *bidder (B)* and the (passive) *participant (P)*. The main interactions are characterized by the following rules:

Join	$join_{SK}(a) := ValidRole(a, \{\sigma\} \cup \{\rho\} \cup \{v\}) \wedge \{\alpha\} = \{\alpha\} \cup a$
Leave	$leave_{SK}(a) := a \in \{\alpha\} \wedge \{\alpha\} = \{\alpha\} - a$
Put	$put_{SK}(a, m') := a \in \{\alpha\} \wedge belongs(m', \overline{I}) \wedge ValidRole(a, \{\sigma\}) \wedge \wedge \rho_m \in \{\alpha\} \wedge \{m\} = \{m\} \cup m'$
Get	$get_{SK}(a) := (a \in \{\alpha\} \wedge \exists m' \in \{m\} \wedge ValidRole(a, \{\rho\}) \wedge \wedge \{m\} = \{m\} - m' \wedge return\ m') \vee (a \in \{\alpha\} \wedge \wedge \exists m' \in \{m\} \wedge ValidRole(a, \{v\}) \wedge return\ m')$
Predicative Get	$p\_get_{SK}(a, P) := (a \in \{\alpha\} \wedge \exists m' \in \{m\} \wedge ValidRole(a, \{\rho\}) \wedge P(m) \wedge \wedge \{m\} = \{m\} - m' \wedge return\ m') \vee (a \in \{\alpha\} \wedge \wedge \exists m' \in \{m\} \wedge ValidRole(a, \{v\}) \wedge P(m) \wedge return\ m')$
Insert	$insert_{CK}(SK_*) := \exists \overline{SK} \in CK = \{SK_1, ..., SK_n\} \wedge spec(SK_*, \overline{SK}) \wedge CK = CK \cup SK_*$
Extract	$extract_{CK}(SK_*) := \exists SK_* \in CK = \{SK_1, ..., SK_n\} \wedge CK = CK - SK_*$

Figure 1. Basic Communicator Primitives and Behaviour

1. For any speech act sent from the auctioneer to the bidders, the real receivers are the bidders, but the virtual ones are both the bidders and the participants.
2. For any speech act sent from a bidder to the auctioneer (a bid), the real receiver is the auctioneer but the virtual set is composed of bidders and participants.
3. A participant cannot make a bid (unless it becomes a bidder).
4. Communication between a bidder and the auctioneer has to be performed using LISP as content language.
5. Communication between a bidder and the auctioneer is subject to the rules of the auction protocol used in the application.
6. Private communication between a bidder or participant and the auctioneer is not allowed.
7. Communication (of any type) between bidders and participants is allowed.

On the basis of the cited rules, we can derive the following interaction classes:

- $\overline{I_1} =< A, Bs, \{A, Bs, Ps\}, *, *, LISP, * >$ (rules 1 and 4)
- $\overline{I_2} =< B, A, \{A, Bs, Ps\}, *, *, LISP, * >$ (rules 2 and 4)
- $\overline{I_3} =< \{Bs, Ps\}, \{Bs, Ps\}, *, *, *, *, * >$ (rule 7)

Please note the use of singular (*B*) and plural (*Bs*) in role names to indicate a specific agent or a set of agents ($\overline{I_2}$ specifies a message sent by a single bidder while $\overline{I_1}$ specifies a message sent to all the bidders). Finally, we can build the Communicator for the interactions for the electronic auction application by grouping together the Simple Communicator defined by $\overline{I_1}$, $\overline{I_2}$ and $\overline{I_3}$. A thus-defined Communicator also satisfies rules 3 and 6, since a message of class $< A, B, B, *, *, *, * >$, $< B, A, A, *, *, *, * >$ or $< P, A, A, *, *, *, * >$is not included[3].

3.2 Design Time and Run Time Interactions

The derived coordination framework is able to handle interactions defined during the analysis and design phase of an agent-based application. However, at run time, in order to perform activities which are collateral, or for the support of the main task of the application, agents can cooperate by exchanging messages which have not been explicitly defined at design time. For example, in the electronic auction application a private interaction between two bidders, Alice and Bob, outside of the context of the auction protocol, could also be possibile: this is a run time interaction which, even if it falls into class $\overline{I_3}$ of the Sect. 3.1, it will not be routed correctly since the virtual receiver set of $\overline{I_3}$ is *any* and each message could be picked by any agent. What we need is the possibility to define, run-time, an interaction class of the type $\overline{I_P} =< \{alice, bob\}, \{alice, bob\}, nil, *, *, *, * >$ and to add it to the Communicator of our auction application. This example states that our model must provide the following additional characteristics: (1) the possibility to specify *agent names* (e.g. Alice, Bob) in addition to *roles*, and (2) the support of a run time reconfiguration of Communicators. The former requirement can be met by allowing the specification of both roles and names in the sets $\{\sigma\}$, $\{\rho\}$ and $\{v\}$, and by modifying the predicate $ValidRole$ defined by 5 as follows:

$$ValidRole(r, P) \Leftarrow (isRole(r) \wedge \exists r' \in P \wedge isRole(r') \wedge Role(r) = r') \vee (\neg isRole(r) \wedge \wedge \exists r' \in P \wedge \neg isRole(r') \wedge r = r')$$

[3] Rule 5 is related to the evolution of the protocol, which is not dealt with by the Communicator model.

The latter requirement needs a new primitive able to add, run time, a new Simple Communicator $SK_* =< \overline{I_*}, \{m\}, \{\alpha\} >$ to an existing CK. This primitive does not have to perform a simple union operation but, in order not to violate the semantics of Communicators, it also has to check that $\overline{I_*}$ *is allowed* in our application, i.e.

$$\exists \overline{I} \in CK : \forall m' \wedge belongs(m', \overline{I_*}) \Rightarrow belongs(m', \overline{I}) \tag{10}$$

In other words, the new interaction class $\overline{I_*}$ *must be a subclass* or a *specialization* of another class still defined by a Simple Communicator of CK. Given this, let us define the *specialization* predicate for both interaction classes and Simple Communicators[4]:

$$\begin{aligned} spec(\overline{I_*}, \overline{I}) \quad \Leftarrow \quad & \forall m' \wedge belongs(m', \overline{I_*}) \wedge \\ & \wedge belongs(m', \overline{I}) \wedge \exists m'' \wedge \\ & \wedge belongs(m'', \overline{I}) \wedge \\ & \wedge \neg belongs(m'', \overline{I_*}) \end{aligned} \tag{11}$$

$$\begin{aligned} spec(SK_*, SK) \quad \Leftarrow \quad & SK_* =< \overline{I_*}, \{m\}_*, \{\alpha\}_* > \wedge \\ & \wedge SK =< \overline{I}, \{m\}, \{\alpha\} > \wedge \\ & \wedge spec(\overline{I_*}, \overline{I}) \end{aligned} \tag{12}$$

Now we are able to introduce two primitives, $insert_{CK}$ and $extract_{CK}$, which allow respectively the insertion of a specialized SK_* into an existing Communicator, and the extraction of an existing SK from a Communicator (Figure 1). Inserting a new SK_* requires the latter to be a specialization of an $\overline{SK} \in CK$, otherwise the coordination rules defined at application design time will be violated. Even if required, this has a collateral effect: since relation 12 is true, an incoming message, to be placed into SK_*, would be placed also in $\overline{SK}$, and this is in contrast with the semantics of the interaction in which the message is exchanged. In fact, let us refer to the auction example: there, interaction class $\overline{I_P}$ is a specialization of $\overline{I_3}$, thus if a private message between Alice and Bob is placed also in the SK of $\overline{I_3}$, it could be picked by any other bidder or participant agent. To avoid this problem, we have to modify the behaviour of put_{CK} by using the following rule: if a message m' belongs to several interaction classes and these classes are in a specialization relationship, then the message must be placed in the SK of the *most specialized* class.

With the introduction of these modifications, the model of Communicators now provides a great flexibility allowing agents to perform interactions according to their requirements, also preserving the semantics of both application design-time interactions and message exchanging in the context of that application.

[4]The predicate is read as "$\overline{I_*}$ is a specialization of $\overline{I}$".

4 Related Work

Our work on Communicators is derived from the concept of *message sharing* provided by the ARCA mobile agent framework [6]. Here, an agent can share its messages (stored in the incoming message queue), matching a given regular expression, with one or more co-located (mobile) agents, thus emulating the concept of real and virtual receivers. The Communicator model generalizes this concept by extending the sharing also to non co-located agents, providing also a more fine-grained programming of the coordination infrastructure.

A large number of coordination models for agents derives from the re-adaptation of the Linda coordination language to a distributed multi-agent environment [2, 1], in order to support the different behaviour of agents with respect to standard processes[5]. They offer some advantages such as the *spatial and temporal uncoupling*, and also provide a good model for knowledge sharing. Even if they are designed to meet agent requirements, the operations provided – reading and writing tuples through **in** and **out** primitives and other variants – *do not fit* a human-like interaction of the agent model. Agents need to handle *agents names* (or *addresses*) rather than tuple space names, and to forward a message, encoded in a speech act, directly to another agent, a set of pre-defined agents, the entire agent community, etc. Anyway, Communicators can emulate Linda-based models but, since they are designed on the basis of the analysis of agent requirements, they are able to better support the human-like behaviour of agent coordination.

An approach similar to the Communicator model is given in [11], where Minsky et al. propose the concept of *Law Governed Interactions* to support "coordination policies" in distributed systems. Here, coordination is mediated by *controllers* which embed the rules and the actions to be performed when a message of a given type is sent or delivered. With respect to Communicators, controllers have *states* and the behaviour is programmed through a sort of finite state machine. Although the proposed approach is valid, in our opinion it greatly reduces the autonomous character of agents, which are really the responsible parties of the evolution of the coordination. In any case, a controller can be emulated, in our model, using a brokering agent which embeds the coordination policies to be met.

Another interesting work which has been considered is that proposed in [12], where the coordination language MANIFOLD is introduced, which provides a *control-driven* coordination model. In MANIFOLD, agents communicate through *streams* and *events*, upon which agents have the complete control (creating/destroying streams, modifying streams and behaviour events , etc.). The language is powerful, but the coordinables are more similar to processes rather

[5]Linda was initially designed to coordinate processes.

than agents, thus it does not fit the human-like behaviour of agents.

5 Conclusions

This paper has presented a coordination model for agents based on an infrastructure with characteristics that directly derive from the agent model and from coordination requirements in a multi-agent application. The basic component, the *Communicator*, is designed to provide primitives and behaviours which fit as closely as possible the human-like character of agents. The paper has also shown how the proposed model is able to provide both direct and indirect communication. The authors are currently working on implementation of a prototype of the model, in order to evaluate the effectiveness and the overhead introduced in a distributed multi-agent application.

References

[1] G. Cabri, L. Leonardi, and F. Zambonelli. Mobile-Agent Coordination Models for Internet Applications. *IEEE Computer*, 33(2), February 2000.

[2] N. Carriero and D. Gelernter. Linda in Context. *Comm. ACM*, 32 - No. 4, April 1989.

[3] P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, and A. Knoche. Coordinating Multiagent Applications on the WWW: A Reference Architecture. *IEEE Transaction on Software Engineering*, 24(5), 1998.

[4] A. Di Stefano, L. Lo Bello, and C. Santoro. Naming and Locating Mobile Agents in an Internet Environment. In *EDOC '99 IEEE Intl. Conference*. Mannheim, Sept 27-30 1999.

[5] A. Di Stefano and C. Santoro. NETCHASER: Agent Support for Personal Mobility. *IEEE Internet Computing*, 4(2), March/April 2000.

[6] A. Di Stefano and C. Santoro. The Coordination Infrastructure of the ARCA framework. In 4^{th} *Intl. Conference on Autonomous Agents*. Barcelona, Spain, June 3-7 2000.

[7] T. Finin and Y. Labour. A Proposal for a New KQML Specification. Technical Report TR-CS-97-03, Computer Science and Electrical Engineering Dept., Univ. of Maryland., 1997.

[8] Foundation for Intelligent Physical Agents. FIPA-ACL Specification, available at http://www.fipa.org.

[9] J. Bradshaw et al., editor. *Software Agents*. AAAI Press, Cambrigde, Mass., 1997.

[10] Y. Labrou, T. Finin, and Y. Peng. Agent Communication Languages: the Current Landscape. *IEEE Intelligent Systems*, March-April 1999.

[11] N. H. Minsky. The Imposition of Protocols over Open Distributed Systems. *IEEE Trans. on Software Engineering*, February 1991.

[12] E. Rutten, F. Arbab, and I. Herman. Formal Specification of Manifold: a preliminary study. Technical Report CS-R9215, CWI, Amsterdam, 1992.

[13] M. Wooldridge, N. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3), 2000.

[14] M. J. Wooldridge. *Multiagent Systems*. G. Weiss, editor. The MIT Press, April 1999.