Context-dependency in Internet-agent Coordination

Giacomo Cabri, Letizia Leonardi, Franco Zambonelli
Dipartimento di Scienze dell'Ingegneria – Università di Modena e Reggio Emilia
Via Campi 213/b – 41100 Modena – ITALY
Phone: +39-059-376735 – Fax: +39-059-376799
{giacomo.cabri, letizia.leonardi, franco.zambonelli}@unimo.it

Abstract

The design and development of Internet applications can take advantage of a paradigm based on autonomous and mobile agents. However, agent mobility introduces peculiar coordination problems in Internet applications. First, it suggests the exploitation of an infrastructure based on a multiplicity of local interaction spaces. Second, it may require coordination activities to be adapted both to the characteristics of the execution environment where they occur and to the needs of the application to which the coordinating agents belong. In this context, this paper introduces the concept of context-dependent coordination based on programmable interaction spaces. On the one hand, interaction spaces associated to different execution environments may be independently programmed so as to lead to differentiated, environment-dependent, behaviors. On the other hand, agents can program the interaction spaces of the visited execution environments to obtain an application-dependent behavior of the interaction spaces themselves. Several examples show how a model of context-dependent coordination can be effectively exploited in Internet applications based on mobile agents. In addition, several systems are briefly presented that, to different extent, define a model of context-dependent coordination.

Keywords*: Internet Applications, Mobile Agents, Locality, Coordination*

Corresponding Author*: Franco Zambonelli*

1. Introduction

The design and development of Internet applications can effectively take advantage of an agent-based approach. By conceiving and developing applications in terms of autonomous entities, capable both of reacting to changes in the environment and of proactively dealing with unexpected situations, one naturally deal with the intrinsic uncertainty and dynamicity of the Internet. In addition, by exploiting the notion of sociality typical of the agent-oriented paradigm, one can more easily decompose an application in terms of a multi-agent organization, where all the necessary interaction protocols find a first-order accommodation and definition.

Among the other characteristics, *agent mobility* [Whi97, KarT98], i.e., the agents' capability of moving across the Internet while executing, is being recognized as a useful property for both the actual execution and the conceptual design of Internet applications. The *physical* movement of a software agent – i.e., the transfer of its code, data, and state – towards the nodes where the resources it needs to access are allocated can provide for saving network bandwidth, and for increasing both the reliability and the efficiency of the execution [Whi97, LanO98]. The *logical* movement of a software agent, instead, refers to the fact that, in agent-based Internet applications, it is suitable to abstract the Internet as a multiplicity of *execution environments* [FugPV98] and to design applications in terms of agents that are aware of the distributed nature of the target and that *logically move* (when not even physically) in different execution environments during their execution.

The presence of mobility, together with the intrinsic openness of the Internet scenario, also introduces several problems in the design and development of Internet applications. Among the others, *coordination* problems, because of the need for agents to interact, communicate, and synchronize their activities, both with other agents (inter-agent coordination) and with resources on the hosting execution environments (agent-environment coordination). With regard to physical mobility, relying only on traditional coordination models (i.e., peer-to-peer and client-server) for global interactions between agents cannot always be feasible if agents can freely move at a world-wide scale [CabLZ00]. With regard to logical mobility, the fact that an agent executes and interacts on different execution environments during its life has several implications on application design. On the one hand, one must take into account that each environment has its specific characteristics and security policies, which may somehow influence and/or constrain the behavior of an agent in

coordinating with it. On the other hand, one may require the coordination activities of application agents to occur according to specific application needs, despite the different characteristics of the local environment and the different coordination rules there enacted.

To overcome the problems introduced by physical mobility of agents, it has already been argued that both inter-agent and agent-environment coordination can effectively rely on an infrastructure based on a multiplicity of independent interactions spaces (whether meeting points or tuple spaces), each associated to an execution environment and in charge of mediating both inter-agent and agent-environment interactions [OmiZ99, CabLZ00]. This makes it possible for agents to coordinate, via the mediation of the local interaction space, without worrying about each other's position and name. In addition, by enhancing interaction spaces with the capability of dynamically programming their behaviors, one can make interaction spaces behave in different ways depending on which agent issued which interaction event. This characteristic can be used to support the logical mobility of agents, by making their coordination *context-dependent*. On the one hand, interaction spaces associated to different execution environments may exhibit different behaviors in response to the same interaction events. This enables to integrate in the form of new behaviors – transparently to agents – whatever needed environment-specific policy to rule the local coordination activities. On the other hand, on a given site, the same interaction events performed by agents belonging to different applications can lead to differentiated, application-dependent, behavior. Thus, agents can carry-on the code needed to implement and control an application-specific coordination policy and install it in the form of a new behavior into the interaction space of the visited site. In this way, agents can coordinate on the site being guaranteed that the interaction space will let them coordinate accordingly to their specific needs.

This paper is organized as follows. Section 2 briefly motivates the adoption of local interaction spaces in Internet applications based on mobile agents, by specifically focussing on tuple-based interaction spaces, and by introducing a simple case study application. Section 3 details the concept of context-dependent coordination and discusses, via several examples, how it can be effectively exploited. Section 4 briefly survey several existing coordination systems and model that, to different extent, tend towards a model of context-dependency. Section 6 concludes the paper, outlining open research directions.

2. From Global and Coupled to Local and Uncoupled Interactions

Physical mobility of agents introduces peculiar problems that discourage the generalized adoption of traditional coordination models, such as peer-to-peer message-passing and client-server, which enforce global and coupled interactions. First, the globality of the Internet scenario and, therefore, of the space of possible interactions, can make communications between distant agents inefficient and unreliable, also calling for complex infrastructure to deal with message forwarding [MurP99], and lookup of agents' positions and names. Second, the intrinsic autonomy and dynamicity of agents clashes with the strict coupling (in terms of both synchronization of the activities and necessity of sharing a common name space) enforced by these models.

The alternative is to make agent coordination rely, whenever possible, on an infrastructure enforcing local and uncoupled interactions. On the one hand, interactions are to be confined within a local interaction space associated to the local execution environment. On the other hand, the local interaction space should act as the medium through which asynchronous and anonymous (i.e., uncoupled) interactions may occur. The mediation of the interaction space can make it possible for agents to indirectly interact with other agents disregarding their current positions and names, and for the local environment to make the interaction space act as the gate through which agent can access the local resources. Although different kinds of infrastructures can be conceived providing this characteristics and inspired by different coordination models (e.g., meetings [Whi97] or event-channels [Bau98]), we specifically suggest the adoption of an infrastructure based on independent tuple spaces [AhuCG86] associated to each execution environment. An agent on a site – via storing and retrieving of tuples in the local tuple space – can exchange data and knowledge messages with other agents, in an associative way, whoever and wherever these agents are, and whenever they will read (have written) the messages. In addition, an environment can publish in the form of tuples in the local space its public resources, to be accessed by locally executing agents.

Let us consider an application in which a mobile searcher agent roams an itinerary of Internet sites to access and analyze specific HTML pages. The searcher agent arrives at a site, retrieves the needed data, and continues its itinerary. However, if the searcher agent discovers the presence of other interesting sites (via HTML links) not originally in its itinerary, it can clone itself and let the clones go to those sites. This can generate a tree of searcher agents, and makes it likely for two agents to arrive in the same site at different times. This situation must be avoided by coordinating

the agents' movements: when on a site, a searcher agent must know if the site has already been visited or not by another searcher agent.

Since a searcher agent not only does not know where the other searcher agents are, but it also ignores who and how many they are, relying on peer-to-peer interaction between them is not feasible: searcher agents are forced to refer to a fixed, well-known, "home agent", acting as mediator for all searcher agents, introducing performance and reliability problems, and also complicating the application design. Instead, a tuple space infrastructure can be exploited in a very simple and elegant way, leading to a more efficient and simple application design. When a searcher agent arrives on a site, it simply checks the local tuple space for the presence of a tuple left by another searcher agent on a previous visit. If the tuple is not found, the site has never been visited and the incoming searcher agent is in charge of leaving a tuple in its turn, to mark its current visit.

In the above example, the tuple space – due to the associative access to tuples – can be effectively used also to access the information about the local HTML files. The administrator can write, in the tuple space, tuples in the form: (filename,extension,keyword,modification_time,size). Searcher agents, by their side, can retrieve information about local HTML files related to the argument "coordination", by asking for tuples matching: (string?,"html","coordination",date?,int?).

3. Towards Context-Dependent Coordination

A model of local and uncoupled interactions not only fits the physical mobility of agents, but also invites thinking applications in terms of logical mobility. When an agent migrates to a new execution environment, its interaction space changes accordingly and is confined within the new environment. In other words, the agent's perceivable world changes as a consequence of its movement.

Even if the interaction model is the same independently of the environment, agents are likely to face several problems in proceeding with their coordination activities in all the different environments they visit. First, coordinating with the agents residing in a foreign environment may require facing all the typical problems of open systems, such as opportunistic behavior in interactions, heterogeneity of languages and protocols [Zam00]. Second, each environment has its specific characteristics and may somehow constrain the behavior of an agent in coordinating with it, for security or resource control reasons. Finally, despite the different characteristics of each

environment, application agents may still require that their coordination occurs according to specific application needs.

In such a scenario, if the interaction space is simply an infrastructure for storing and forwarding of data and message, agents have to embed in their code all the intelligence needed to effectively handle their coordination activities in foreign execution environments. Alternatively, one can think about having the interaction space itself in charge of the burden of handling all coordination activities, to meet either environment-specific needs, or application-specific ones, or both. However, this requires the interaction space to be somehow active and to enable a dynamic adaptation of its behavior in response to interaction events.

For instance, one can enhance the basic tuple-space model towards a *programmable tuple space model*: for any kind of access events, a new behavior in response to it can be programmed to override the basic behavior of the tuple space (i.e., storing and extraction of tuples based on a stateless and built-in pattern-matching mechanism). To this end, one must: *(i)* characterize the kind of access events of interest, in terms the identity of the agent performing it, the primitive used to access the tuple space, and the parameter tuple supplied in the primitive; *(ii)* express the new behavior (also called *reaction*) to be assumed by the tuple space in response to this kind of access events. Similar considerations may apply when considering the enhancement towards programmability of interaction spaces other than those based on the tuple space model

The adoption of a programmable tuple space model in mobile-agent Internet applications, as well as of any model based on programmable and local interaction spaces, naturally leads to a model of *context-dependent coordination*. As discussed in the following of this section with reference to programmable tuple spaces, this enables: *(i)* the administrator to adapt the behavior of the interaction space to enforce – transparently to agents – site-specific policies on the local interactions; *(ii)* application agents to dynamically adapt the behavior of the interaction space accordingly to specific application needs.

3.1 Environment-Dependent Coordination

Agents, while accessing a tuple space always with the same interface, can have the semantics of their coordination activities (as well as their perception of the environment) affected by the specific behavior programmed for the local tuple space. In other words, the same interactions can have different effects depending on the site in which they are performed. This can be used both to embed

security policies in the tuple space, and to help agents entering the new environment without having to explicitly take in charge of all the issues implied in executing in and accessing a foreign environment.

Enforcing Security. When a site opens itself to the execution of mobile agents, it must be aware that malicious agents are likely to arrive at the site and undermine the integrity of its data and resources, which must be therefore protected from unauthorized accesses. Agents, on their side, may not be aware of the security policies adopted by an execution environment, so that their interactions with an environment are likely to issue a large number of security exceptions, to be handled by the agents themselves. However, if all the interactions with the environment are mediated by a programmable tuple space, the above problems can find a simple and elegant solution. In fact, the administrator can program the behavior of the tuple space so as to make any unauthorized access to the tuple space harmless without requiring the agent to handle exceptions. In the application example, if a searcher agent tries to extract a tuple representing a HTML file (while it has only the right to read that kind of tuples) a properly programmed tuple space can provide that tuple to agent without actually extracting it from the tuple space. As an additional example, a stateful reaction can be programmed to let searcher agents read only a specified amount of HTML tuples: once the specified amount has been extracted, the tuple space simply starts appearing to agents as if it does not contain further HTML tuples. In both examples, agents do not experience any exception when violating the access control policies of the environment: they simply perceive a different view of the tuple space content, while the local environment can maintain and enforce its security policies.

Handling Heterogeneity. Different execution environments can adopt different choices w.r.t. the format of their data and resources stored, as well as w.r.t. their representation in the local tuple space. Agents, in these cases, have to explicitly deal with heterogeneity and somehow discover how to fruitfully access the environment. When adopting a programmable tuple space model, a completely different perspective can be adopted. In particular, a tuple space can be programmed so as to react to the access performed by agents by somehow transforming the agents' request and making them homogeneous to the local representation. Therefore, on the one hand an agent can perceive a tuple space as if it were homogeneous to its expectations; on the other hand, the environment is not forced to change its tuple-based representation of the environment or duplicate it in different formats. In the application example, searcher agents look on a site for HTML files

having the “html” extension, by asking for the corresponding HTML tuples. If the environment, by its side, stores HTML pages in files having the “htm” extensions, and shapes its tuples accordingly, a searcher agent, unless intelligent enough, has no possibility of discovering the presence HTML pages of that site. Then, the administrator (which is supposed to know that most of the agents will look for “html” files) can modify the behavior of the tuple space so as to transform the agents’ requests into requests for “htm” files, transparently to agents. In other words, the administrator modifies the matching mechanisms so as to make “html” match with “htm” in HTML tuples.

Supporting Open Interactions. Exploiting tuple space programmability to deal with heterogeneity issues naturally extends also to these cases in which a site is supposed to be open to host the execution of agents belonging to different applications and organizations, possibly heterogeneous in terms of supported languages and protocols, and nevertheless in need of coordinating with each other. Again, a tuple space can be used as a mediator, so as to support the coordination activities among a group of heterogeneous agents. Even more, a trusted site can be used to act as an impartial arbiter, in charge of controlling the interactions of agents with other, possibly self-interested, agents, that would be likely to be unfruitful or damaging the agents, otherwise. As an example still related to the information retrieval area, the tuple space can be used so as to control the behavior of opportunistic “advertising agents” that could have interest in diverting the research of searcher agents toward specific commercial sites.

3.2 Application-Dependent Coordination

Application designers can exploit the programmability of the tuple spaces in different ways. It can be used to facilitate the access to the information on the visited site, to support the exchange of complex knowledge between agents and to implement complex coordination protocols. More generally, application designers can exploit programmability of tuple spaces so as to adapt the interaction model to their specific, application-dependent, needs. Of course, since these modifications of the tuple spaces’ behaviors are intended to meet specific application needs, some form of confinement must be provided by the tuple spaces to ensure that any application-specific behavior can affect only those access events performed by agents of that specific application.

Facilitating Access. Let us consider again the application example. To avoid duplicated work, one can think about having searcher agents put a "marker" tuple in the space of the visited sites. This makes later incoming searcher agents of the same application aware of the fact that the site has

already been visited. Even better, the marker tuple can report the time of the visit: later incoming searcher agents, in this case, can simply detect which files have been modified since the previous visit and collect only the information that is updated w.r.t. that held by the earlier visiting agent. However, the basic pattern matching mechanism would force a searcher agent to retrieve all the tuples representing files and, subsequently, to select only those representing files which have been modified since the last visit. Therefore, the application calls for a different pattern-matching mechanism, to effectively support its inter-agent interaction needs, i.e., one that can make it possible to select all HTML tuples in which the modification_time field is greater than the one specified in the template tuple. Of course, only those access events performed by agents of the application example must trigger the reaction leading to the new pattern-matching mechanism. That is, the specific pattern-matching mechanism has to influence only that application context.

Exchanging Knowledge. In the application example, putting a marker tuple in the space to be read by other application agents is a sort of knowledge exchange between agents of the same applications: "I have visited this site at time T and, if any agent ever reads that tuple, has to know that it has to analyze only those HTML files that has been modified since time T". However, when we designed the application, we already knew that the application was composed of multiple agents, which were likely to arrive on the same site at different times. Therefore, we designed each agent so as to explicitly look for marker tuples in the visited site. Let us now suppose, instead, that an agent does not know that there are other application agents that can possibly have visited the same site in the past. In this case, of course, an agent does not worry at all about acquiring this knowledge by checking for the presence of a marker tuple in the tuple space. Actually, this is a very common case in multi-agent applications: an agent acquires some sorts of new knowledge or expertise (“I know what information was in that site at time X”) that can be very useful to the other agents to improve their work (“If I go to that site I have to analyze updated information only”) and other agents, instead, cannot think at acquiring (“Why should I suppose that someone else has already visited a site?”). In the presence of a programmable tuple space model, the knowledge agents acquire can, in several cases, be embedded in the form of new tuple space behavior, that is meant to influence further accesses to the tuple space performed by other agents of the same application. In other words, an agent can install in a tuple space a reaction that reflects, in term of produced information, the knowledge acquired, to be transferred to other application agents. Via the reaction, other

application agents can exploit this knowledge in a transparent way, without having to be explicitly informed and without being forced to ask for this knowledge. With regard to the above examples, an agent that has visited a site can install a reaction that returns, to other agents of the same application, only those files that have been modified since its last visit. In that way, another agent arriving there does not have to know or to worry about previous visits, because the reaction guarantees the avoidance of duplicated work. The above example shows how making coordination on a site application-dependent can lead to a very powerful model, which is likely to simplify inter-agent coordination and to enable dynamic exchange of knowledge without influencing at all the agent code.

Implementing Coordination Protocols. The burden of implementing a coordination protocol, whether involving agents interacting in a peer-to-peer way or indirectly via a not-programmable tuple space, increases the complexity of agents. In fact, agents are in charge of implementing in their code the capability of directly handling and controlling the proper execution of the coordination protocols. A better and cleaner solution would be to charge the interaction space itself (i.e., the tuple space) of the burden of controlling the coordination protocols. In particular, a programmable tuple space can express the control part of the coordination protocols in terms of behaviors in response to access events. This can free agents from the duty of actually controlling the execution of the protocol, and provides for a clean separation of concerns between the "computational" task of application agents and the "coordination rules" required for the protocols of the application.

4. Context-Dependent Coordination in Existing Systems

A few of the proposals in the area of mobile agents explicitly focus on coordination models suited to mobility. Also, to our knowledge, none of these proposals explicitly introduce concepts somehow related to the one of context-dependent coordination.

MARS (*M*obile *A*gent *R*eactive *S*paces), developed at the University of Modena and Reggio Emilia, and firstly described in [CabLZ98], implements a portable and programmable Linda-like coordination architecture for Java mobile agents, which naturally supports a context-dependent coordination model. Globally, the MARS architecture is made up of a multiplicity of independent programmable tuple spaces, each one associated to a node and accessed by (and only by) the agents locally executing in that node: in other words, in MARS, logical mobility of application agents has

to necessarily imply their physical mobility. MARS tuple spaces can be independently programmed by associating reactions to access events, via *meta-level tuples* stored in a *meta-level* tuple space. A meta-level tuple represents a specific event (or class of access events) in terms of identity of the invoking agents, type of operation performed, type of tuple/template provided. When an access event performed on the base level MARS tuple spaces matches the meta-level tuple, the execution of a method of a specified Java object is triggered to replace the basic pattern-matching mechanism. Several meta-tuples can match a single access event, in which case all the corresponding reactions are composed and executed. When a reaction method executes, it is provided with the information about the identity of the agents that has triggered the reaction, the operation it has performed, as well as the output of the matching mechanism (or of the previously executed reaction in the pipeline). The fact that a reaction can be associated to access events either when performed by agents with a specific identity or independently of the identity of the accessing agents enables for both environment-dependent coordination (the reaction applies to all agents), and application-dependent one (the reaction applies only to specific agents).

The *TuCSoN* model [OmiZ99], developed in the context of a research project affiliated to MARS, adopts a very similar architectural model, and enhances it by making it possible for agents to refer to remote tuple spaces via URLs, as an Internet service. This enforces network-awareness and logical mobility without forcing physical agent mobility. With regard to the tuple space model, TuCSoN resembles MARS in its full programming capability of tuple spaces, although TuCSoN it defines logic tuple spaces where both tuples and tuple space behaviors are expressed in terms of untyped first-order logic terms, and where unification is the basic pattern-matching mechanism.

The LIME model attempts to face the problem of handling interactions in the presence of mobility in a more general way, also including mobile users and devices [PicMR99]. Each “agent”, whether a software agent, an Internet node, or a physical mobile device, owns and carries on in its movement a tuple space. The agent-owned tuple space is the only medium provided to the agent itself for interactions. Whenever the agent keeps in touch (i.e., gets connected) with another agent, the tuple spaces owned by each of the agents merge together, thus allowing to interact via the merged tuple space. A limited form of reactivity is provided to automatically move tuples across tuple spaces, and to notify agents about the events occurring in their tuple space. Although not explicitly mentioned by their designers, LIME naturally promotes a primitive form of context-

dependent coordination: the effect of an agent accessing to its own-tuple space can be very different depending on whether the tuple space is currently merged with other tuple spaces and which ones, that is, depending on which context the agent currently belongs to.

The model of Law-governed interactions, described in [MinU00], addresses the problem of making peer-to-peer coordination within a group of non-mobile agents obey to a set of specified rules. An agent can initiate a group, and fix the rules to be respected by other agents willing to enter and interact within the group. These rules are typically security-oriented, and express which operations the agents are allowed to perform, and in which order. To enforce these rules, the model dynamically associates a controller process to each agent that joins the group. The controller process intercepts all messages to from the agent and, by coordinating with other controllers, checks their compatibility with the enforced rules. Although the Law-governed interaction model does not address in any way the problems related to agent mobility and does not explicitly identify environment-dependent and application-dependent coordination, it has the full potential for enforcing context-dependent coordination in complex systems of non-mobile agents interacting in a peer-to-peer way.

As a final note, it is interesting to note that the multi-agent systems community is recently recognizing that interactions in agent systems can no longer simply rely on the agent capability of communicating via agent communication languages and of acting accordingly to the need/expectations of each agent in the system. Instead, concepts such as “organizational rules” and “social laws” are receiving more and more attention [MosT95, DemC96, FerG98], and are leading to a variety of models and proposals. However, these concepts and models often lack of concrete and efficient implementations. The concept of context-dependent coordination introduced by this paper is likely to facilitate the definition and the implementation of complex multi-agent systems on the Internet, in that concepts such as organizational rules and social laws can be easily enforced by tuple space programming.

5. Conclusions and Work in Progress

In this paper, we have introduced the concept of context-dependent coordination for mobile agents, and have shown how this concept can be effectively exploited in Internet applications. Defining a framework in which agent coordination in the Internet may easily be represented and controlled, and

may be tuned to the specific needs of both applications and environments, can simplify application design and management, and may represent an important driving force towards the successful deployment of Internet applications based on mobile agents.

Several issues not addressed by this paper still needs to be analyzed for the proposed concepts to be effectively usable and used. In particular: *(i)* coordination infrastructures must be somehow integrated with Agent Communication Languages [Fin94], to let agents interact in terms of high-level, knowledge-based, information; *(ii)* roles and role models [Ken99], due to their importance in the analysis and design of multi-agent applications, have to be somehow integrated in the model; *(iii)* specific software-engineering methodologies [Zam00] must be defined to help in developing agent-based applications and in clearly devising whether and how to exploit agent mobility and context-dependency.

All the above issues are the current interest of our research group. In addition, we aim at defining a more general and formal notion of context-dependent coordination, possibly exploiting already defined formal models for mobile systems [CarG98].

References

[AhuCG86] S. Ahuja, N. Carriero, D. Gelernter, "Linda and Friends", IEEE Computer, Vol. 19, No. 8, pp. 26-34, August 1986.

[Bau98] J. Baumann, F. Hohl, K. Rothermel, M. Straßer, "Mole - Concepts of a Mobile Agent System", The World Wide Web Journal, Vol. 1, No. 3, pp. 123-137, 1998.

[CabLZ98] G. Cabri, L. Leonardi, F. Zambonelli, "Reactive Tuple Spaces for Mobile Agent Coordination", 2nd Workshop on Mobile Agents, LNCS No. 1477, Sept. 1998.

[CabLZ00] G. Cabri, L. Leonardi, F. Zambonelli, "Mobile-Agent Coordination Models for Internet Applications ", IEEE Computer, Vol. 33, No. 2, pp. 82-89, Feb. 2000.

[CarG98] L. Cardelli, D. Gordon, "Mobile Ambients", Foundations of Software Science and Computational Structures, LNCS No. 1378, pp. 140-155, 1998.

[DemC96] Y. Demazeau, A.C. Rocha Costa, "Populations and Organizations in Open Multi-Agent Systems", 1st National Symposium on Parallel and Distributed Artificial Intelligence", 1996.

[DenNO98] E. Denti, A. Natali, A. Omicini, "On the Expressive Power of a Language for Programmable Coordination Media", Proceedings of the ACM Symposium on Applied Computing, ACM, 1998.

[Fin94] T. Finin at al., "KQML as an Agent Communication Language", 3rd International Conference on Information Knowledge and Management", November 1994.

[FerG98] J. Ferber, O. Gutknecht, "A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems", 3rd International Conference on Multi-Agent Systems, Paris (F), July 1998, IEEE CS Press, pp. 128-135.

[FugPV98] A. Fuggetta, G. Picco, G. Vigna, "Understanding Code Mobility", IEEE Transactions on

Software Engineering, Vol. 24, No. 5, pp. 352-361, May 1998.

[Gra98] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus, "D'Agents: Security in a Multiple-Language Mobile-agent System ", Mobile Agents and Security, Lecture Notes in Computer Science, No. 1419, pages 154-187, Springer-Verlag, 1998.

[KarT98] N. M. Karnik, A. R. Tripathi, "Design Issues in Mobile-Agent Programming Systems", IEEE Concurrency, Vol. 6, No. 3, pp. 52-61, July-September 1998.

[Ken99] E. Kendall, "Role Modelling for Agent Systems Analysis, Design and Implementation", 1st International Symposium on Agent Systems and Applications", Palm Springs (CA), IEEE CS Press, Oct. 1999.

[LanO98] D. B. Lange, M. Oshima, "Programming and Deploying Java™ Mobile Agents with Aglets™", Addison-Wesley, Reading (MA), August 1998.

[MinU00] N.H. Minky, V. Ungureanu, "Law-Governed Interaction: A Coordination & Control Mechanism for Heterogeneous Distributed Systems", Draft Technical Report, Department of Computer Science, Rutgers University, 2000, available at http://www.cs.rutgers.edu/~minsky/pubs.html.

[MosT95] Y. Moses, M. Tenneholtz, "Artificial Social Systems", Computers and Artificial Intelligence, Vol. 14, No. 3, pp. 533-562, 1995.

[MurP99] A.L. Murphy, G.P. Picco, "Reliable Communications for Highly-Mobile Agents", 1st International Symposium on Agent Systems and Applications", Palm Springs (CA), IEEE CS Press, Oct. 1999.

[OmiZ99] A. Omicini, F. Zambonelli, "Coordination for Internet Application Development", Journal of Autonomous Agents and Multi-Agent Systems, Vol. 2, No. 3, pp. 251-269, Sept. 1999.

[PicMR99] G.P. Picco, A.M. Murphy, G.-C. Roman, "LIME: Linda Meets Mobility, 1999 International Conference on Software Engineering, Los Angeles (CA), ACM Press, 1999.

[Whi97] J. White, "Mobile Agents", in J. Bradshaw ed.: Software Agents, AAAI Press, Menlo Park (CA), pp. 437-472, 1997.

[Zam00] F. Zambonelli, N. R. Jennings, A. Omicini, M. J. Wooldridge, "Agent-Oriented Software Engineering for Internet Applications", in Coordination of Internet Agents: Models, Technologies and Applications, Springer, 2000, to appear.