RESEARCH FEATURE

Mobile-Agent Coordination Models for Internet Applications

Internet applications face challenges that mobile agents and the adoption of enhanced coordination models may overcome. To win the race, coordination models have to be inspired by the Linda coordination model and enriched by programmable reactivity.

Giacomo Cabri

Letizia Leonardi

Franco Zambonelli

University of Modena and Reggio Emilia

Each year more applications shift from intranets to the Internet, and Internet-oriented applications become more popular. To fully harness the Web's advantages, we must develop new design and programming paradigms. Traditional distributed applications assign a set of processes to a given execution environment that, acting as local-resource managers, cooperate in a network-unaware fashion. In contrast, the mobile-agent paradigm defines applications as consisting of network-aware entities—agents—which can exhibit mobility by actively changing their execution environment, transferring themselves during execution.[1]

Thus, mobile agents offer several advantages over traditional approaches to Internet applications. They can

- save significant bandwidth by moving locally to the resources they need;
- carry the code to manage remote resources and do not need the remote availability of a specific server;
- proceed without continuous network connections, because interacting entities can be moved to the same site when connections are available, and can then interact without requiring further network connections; and
- work with mobile computing systems.

Although mobile agents have recently attracted widespread interest, several development obstacles remain. The technology still lacks secure and efficient execution support, standardization, appropriate programming languages, and coordination models.[2] Coordination—both between agents and between other entities that agents can encounter during execution in the hosting environments—plays a fundamental role in mobile-agent applications.

We propose a taxonomy of possible coordination models for mobile-agent applications, then use our taxonomy to survey and analyze recent mobile-agent coordination proposals. Our case study, which focuses on a Web-based information-retrieval application, helps show that the mobility of application components and the distribution area's breadth imply coordination problems different from those concerning traditional distributed applications.

In particular, we evaluate which coordination models have the necessary characteristics to suit both mobility and the Internet scenario. Suitable models include coordination paradigms based on fully uncoupling the interacting entities, such as Linda,[3,4] possibly enhanced with the ability to program specific reactions in response to interaction events.[5] Adopting such models can lead to simpler application designs than those derived from traditional coordination models.

CASE STUDY APPLICATION

Our case study involves an information retrieval application. We start with the assumptions that HTML pages spread over the Internet store information and that a given user wants to find all HTML pages that include one specific keyword. The user knows one site—the search's starting point—at which can be found documents that match the keyword. The

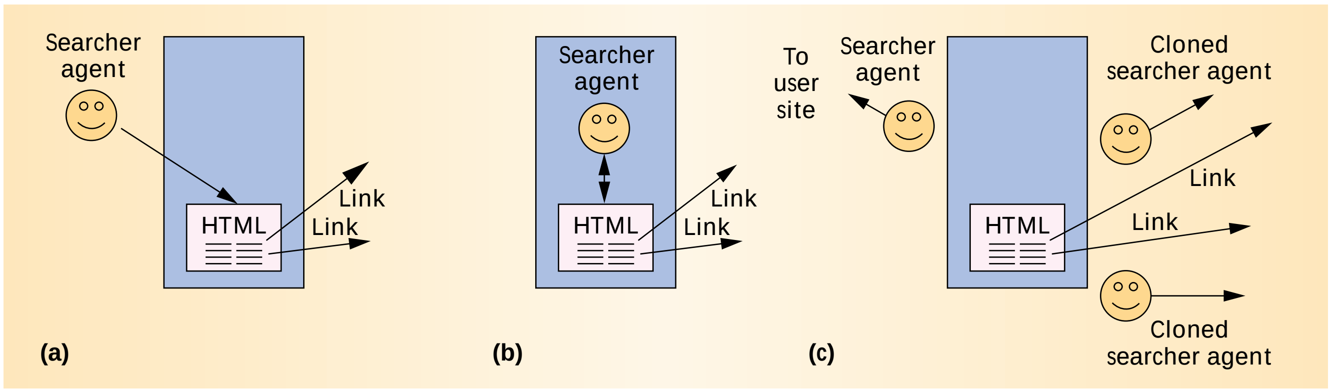


Figure 1. (a) An agent arrives at a site, (b) accesses the local HTML pages, and, before it returns to the user site, (c) creates a new agent for each link to be followed.

user will find other sites containing this information by following the links contained in the first site's HTML pages.

A traditional retrieval design might be based on defining a user process that asks the starting Web server for a page, retrieves and analyzes it, then repeats the same task for each Web server linked to documents that contain the user's keyword. This approach is inefficient, though, because the server must transfer large amounts of data over the network, wasting bandwidth and time. Further, the lack of network-connection reliability will likely cause some site accesses to fail, requiring repeated attempts to analyze some of those sites' documents.

The mobile-agent design, on the other hand, might be based on searcher agents that move locally to a site, analyze the stored documents, then follow the links by cloning themselves. The user could begin this process by creating an initial searcher agent and sending it to the starting site. The agent would then analyze the HTML pages there and return the URLs of those pages containing the user's desired keyword. If the first agent finds documents that contain both the keyword and links to other sites, the agent clones itself once for each link found. As Figure 1 shows, cloning creates new agents with the same code, each one following its assigned link and eventually returning to the home site with the information found. This approach is more efficient and reliable than traditional retrieval methods because it

- saves bandwidth by moving only aggregated data over the Internet and avoiding the transfer of large amounts of data; and
- does not require continuous network connections, because analysis of a site's documents can proceed even in the absence of network connections.

The mobile-agent design must be completed by identifying the coordination needs of the application components and meeting those needs with an appropriate coordination model.

COORDINATION MODEL TAXONOMY

In its nomadic life, a mobile agent must coordinate its activities with other entities, both other mobile agents and resources in the hosting environments. In particular,

- an application may consist of several mobile agents that cooperatively perform a task and must thus exchange data and knowledge, and
- a mobile agent usually must roam through the Internet to access remote resources and services allocated on some network nodes.

In our case study application, we found that HTML page cross-references can cause searcher agents to visit the same sites several times, thus duplicating searches and wasting resources. We used interagent coordination to avoid these multiple visits. Agent-to-execution-environment coordination aims to define a precise protocol for retrieving site information. Letting an agent directly access the local HTML pages is not viable, however, because agents may have no knowledge of the local file system's structure. Further, direct access opens security loopholes.

Extensive study of coordination issues[3] has led to the definition of different coordination-model taxonomies.[4] To analyze the effects of different coordination models in mobile-agent applications, we propose a new taxonomy based on the degrees of spatial and temporal coupling induced by a coordination model:

- Spatially coupled coordination models require that the interacting entities share a common name space; conversely, spatially uncoupled models enforce anonymous interactions.
- Temporally coupled coordination models imply synchronization of the entities involved; conversely, temporally uncoupled coordination models achieve asynchronous interactions.

We can derive four main coordination-model categories from the combinations of these characteristics,

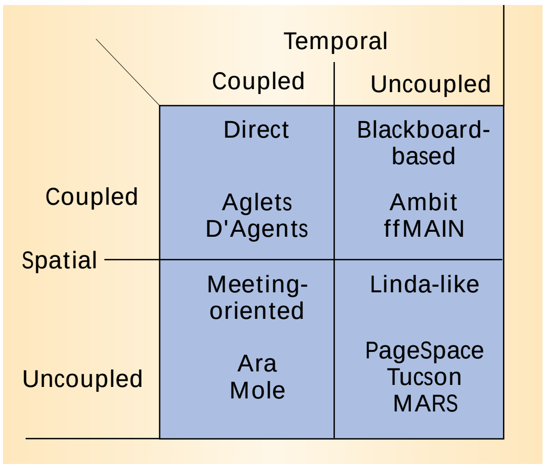


Figure 2. Coordination models for mobile agent applications.

as shown in Figure 2: direct, meeting-oriented, blackboard-based, and Linda-like.

Direct coordination

In direct coordination models, agents use spatial coupling start a communication by explicitly naming the partners involved. In the case of interagent coordination, two agents must agree on a communication protocol, typically peer-to-peer. Access to local resources generally uses client-server coordination, since a hosting environment usually provides local servers for its resource management. Direct coordination usually implies temporal coupling—synchronizing the entities involved.

Most Java-based agent systems—like Aglets and D'Agents[1]—adopt the direct coordination model. At a high level, such systems can exploit the client-server mechanisms typical of the object-oriented paradigm—even remote objects can coordinate through Java remote message invocation. At a low level, these systems can directly exploit TCP/IP, which, while flexible, requires precise definition of a message-exchange protocol.

Direct coordination models do not work as well with Internet applications. If two mobile agents communicate directly on a wide scale, they must be localized by means of complex and highly informed routing protocols. Further, repeated interactions require stable network connections, which makes communication highly dependent on network reliability and may lead to failure or unpredictable delays. Finally, dynamic agent creation makes it difficult to adopt a spatially coupled model that requires identifying communication partners.

Middleware systems, such as those that rely on CORBA and DCOM, can solve the problems related to agent location and naming. However, middleware systems are mostly oriented to allow interoperability among heterogeneous software components rather than to define a globally distributed computing environment. Thus, they still rely on direct communications, based on remote procedure calls, for their services. Although middleware systems can facilitate the use of independently developed components, using them to enable direct mobile-agent coordination risks causes latency and reliability problems.

In our case study application, agents cannot know how many other agents the application contains, because the application creates searcher agents dynamically, depending on the links found. Thus, choosing a direct coordination model forces odd design choices: Ad hoc application entities must be introduced to avoid multiple visits to the same site. As an example, a fixed entity on a given site—say, the user's site—can assume the communication server role shown in Figure 3. Whenever a searcher agent arrives at a site, it queries the communication server to find if the site has already been visited. If it has been, the searcher agent can simply terminate without investigating the site further. Otherwise, it informs the communication server to mark the site as visited.

Unfortunately, this solution produces high network traffic, makes the communication server a bottleneck, and requires stable network connections, thus nullifying most mobile-agent advantages. When using mobile agents, direct coordination models can be effectively exploited only for agent-to-local environment interactions, by providing local servers at each site.

Meeting-oriented coordination

This approach aims to define spatially uncoupled models, which let agents interact in the context of meetings without needing to explicitly name the partners involved. Agents join either explicitly or implicitly known meeting points; afterward, they can communicate and synchronize with the other agents that participate in such meetings. Ever-open meetings abstract the role of servers in an execution environment; application agents can open further meeting points as needed. To avoid the problems related to nonlocal communication, such as unpredictable delay and unreliability, meetings often take place in a given execution environment, which allows only local agents to participate.

The Ara mobile-agent system implements a typical example of meeting-oriented coordination.[6] The concept of event-based communication and synchronization, defined by the Object Management Group and implemented in the Mole mobile-agent system,[7] offers a sophisticated form of meeting-oriented coordination. Specific synchronization objects, which agents must share the reference to, assume the role of meetings. Accessing one of these synchronization objects allows agents to implicitly join the meeting.

Meeting-oriented coordination models partially solve the problem of exactly identifying the partners involved,

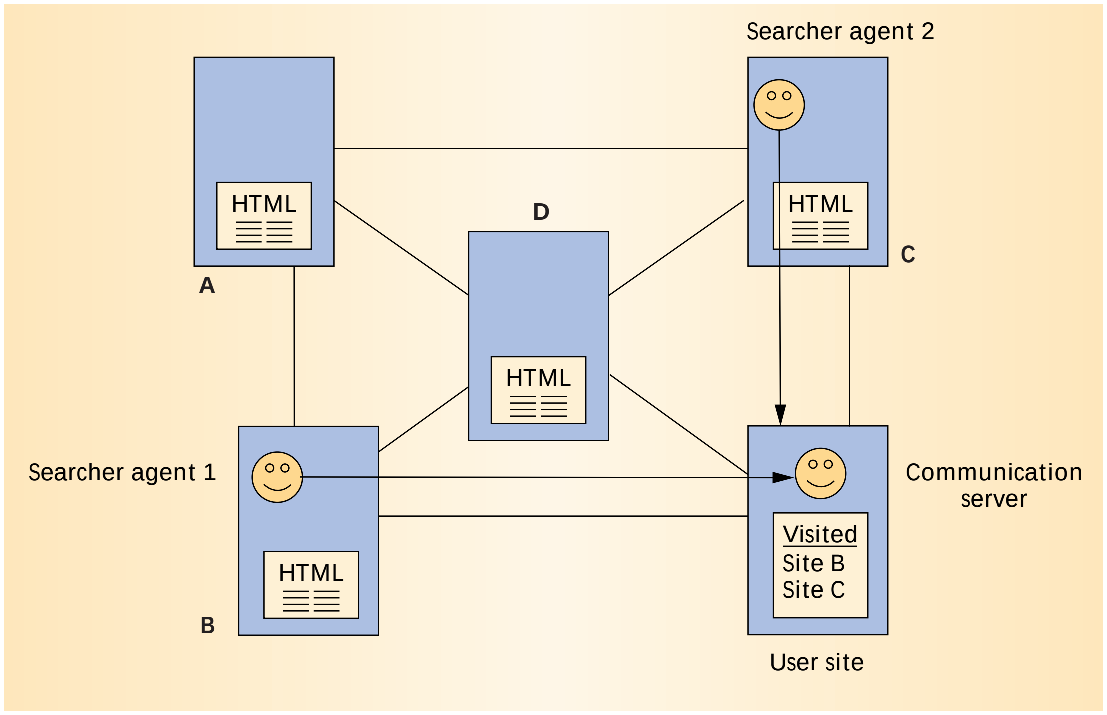


Figure 3. Searcher agents 1 and 2 query the user site's communication server, which informs them that sites B and C have already been visited. This knowledge lets the two searcher agents terminate so that they don't duplicate the effort of other agents.

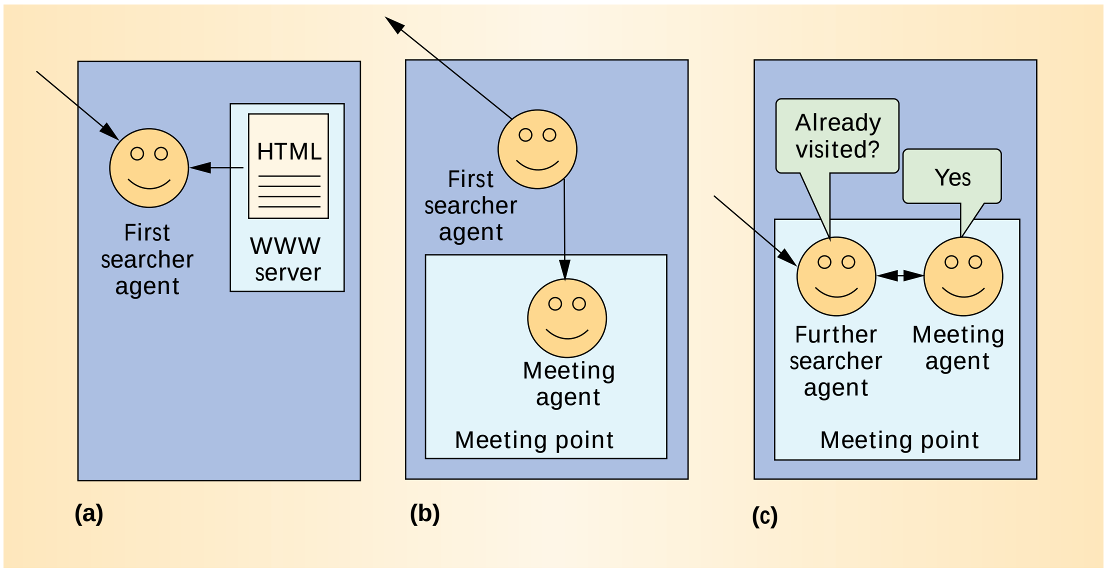


Figure 4. (a) The first searcher agent arrives at a site and retrieves documents containing the user's keyword, then (b) leaves a meeting agent that (c) will notify further incoming agents about the previous visit.

although they cannot achieve the anonymity of full spatial uncoupling: Agents must share at least the common knowledge of the meeting names. These models' major drawback, however, derives from their enforcing synchronization among interacting agents. Since the schedule and the position of agents cannot be predicted in many applications, the models run a high risk of missing interactions. Further, if the meetings are not locally constrained, they must be implemented by message passing, thus inheriting the efficiency and reliability problems of direct coordination models.

In our case study application, this coordination model can most easily be exploited via a locally constrained meeting, introducing an additional agent for each site visited. When a searcher agent has explored a site, it creates a meeting agent before terminating its task. The application forces the meeting agent to reside on the creation site and to enter a predefined meeting point: As Figure 4 shows, every time any other searcher agent arrives at this site, it locally enters the predefined meeting to check for the presence of the meeting agent and then to detect whether or not the site has already been visited. With regard to agent-to-local environment interactions, ever-open meetings can assume the local Web servers' role of retrieving information about a site.

The meeting-oriented design solution produces less network traffic than do direct coordination models and is fully distributed. Nevertheless, the solution forces one meeting agent to remain at each site visited, which leads to two drawbacks:

- A malicious agent can exploit its time on site to send private information to the outside, and

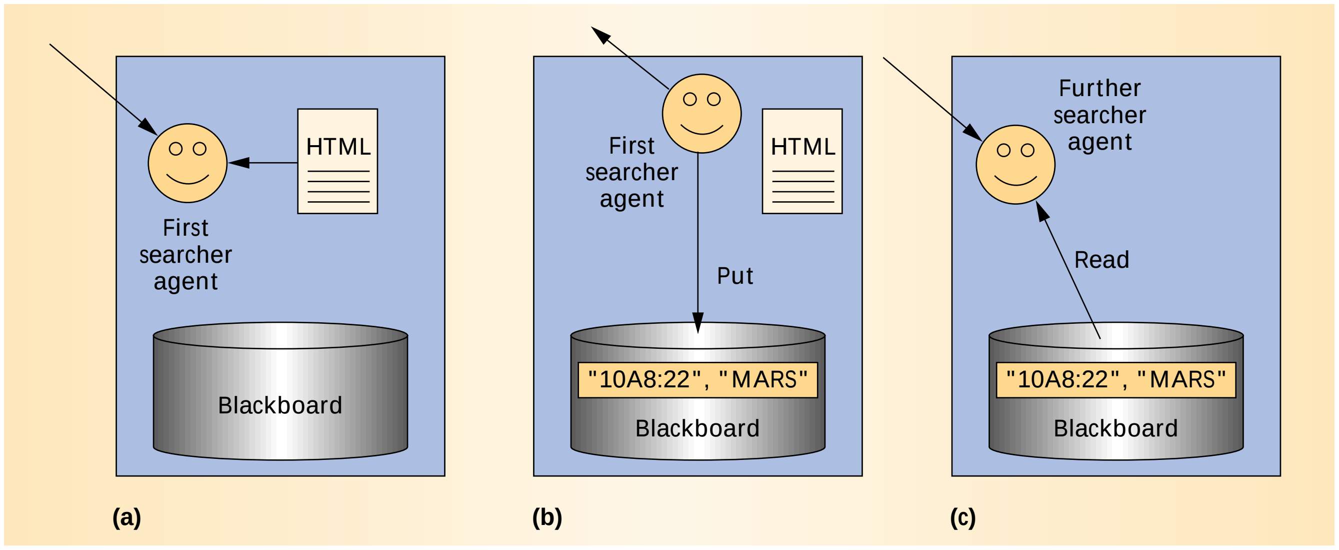


Figure 5. The first searcher agent (a) arrives at a site and retrieves all documents containing the user's keyword. It then leaves a marker message (b) on the blackboard, which is read by further searcher agents (c).

- the meeting agent cannot possibly know when to terminate, unless the system provides global garbage collection mechanisms.

Blackboard-based coordination

In this model type, agents interact via shared data spaces, using them as common repositories to store and retrieve messages. In this sense, interactions are fully temporally uncoupled, but, because agents must agree on a common message identifier to communicate and exchange data via a blackboard, they remain spatially coupled. To overcome scalability problems, a local blackboard can be associated with each hosting environment.

Several systems propose and implement blackboard-based coordination models for mobile-agent applications. In Ambit,[8] a formal model for mobile computations, agents can attach messages to a blackboard on a given site; another agent can retrieve and read these messages when arriving at the same site. The ffMAIN (Frankfurt Mobile Agent Infrastructure**)** agent system[9] defines mobile agents that interact—both with each other and with local resources—via an information space accessed through HTTP. In this space, data—identified via URLs—can be stored, read, and extracted.

Messages can be left on blackboards, no matter where the corresponding receivers are or when they read messages. This temporal uncoupling suits a mobile scenario in which the position and scheduling of the agents cannot be either easily monitored or granted. Further, since all interagent interactions must be performed via a blackboard, hosting environments can easily monitor and control them, thus leading to a more secure execution model than possible with previously mentioned coordination models.

Our case study application could exploit the blackboard model to avoid multiple visits. When a searcher agent arrives at a site, it checks the blackboard for a marker message, to detect whether another agent of the same application looking for the same keyword has already visited the site. Such a message could have the form (application_id, search_keyword). If the agent finds the message, it notices that the site has already been visited and it terminates; otherwise, it must leave a marker message on the blackboard, as shown in Figure 5.

With regard to agent-to-local environment interactions, a blackboard can be exploited to let agents retrieve the information needed without requiring the presence of specialized resource managers. The blackboard also lets the local environment provide all the data it wants to publish while protecting private data. In our application example, the local environment can provide the pathnames of all its publicly accessible files in the form of messages, without using a Web server. However, because of spatial coupling, the blackboard cannot be effectively used to access information about the local HTML pages. There is no way for an agent to retrieve only the HTML files' path names: It is bound to read all messages and successively select only those that correspond to HTML files.

Linda-like coordination

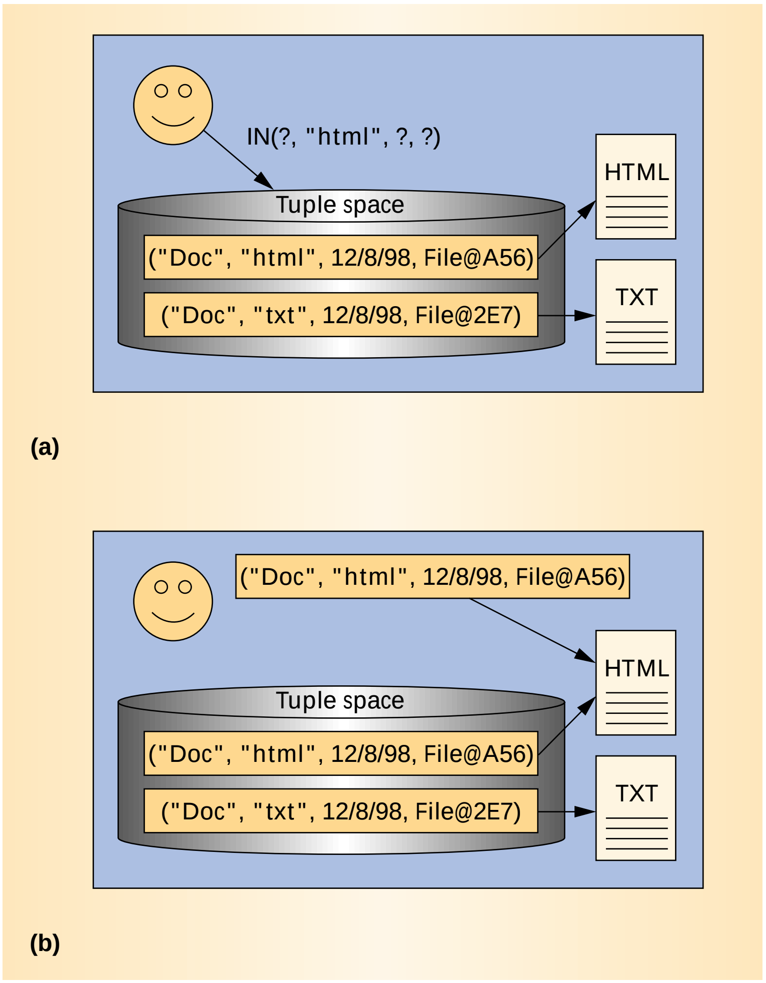
This model type uses local tuple spaces as message containers similar to blackboards. In addition, a tuple space bases its access on associative mechanisms.[3] The system organizes information in tuples and retrieves it using associative pattern-matching. This approach enforces full uncoupling, requiring neither temporal nor spatial agreement.

The PageSpace coordination architecture for interactive Web applications[10] adopts the concept of an associative blackboard. PageSpace lets both mobile and fixed agents use a multiplicity of distributed object-based tuple spaces to store and retrieve object references. Other systems, such as TuCSoN[12] (Tuple Centres Spread over Networks) and MARS[5] (Mobile Agent Reactive Spaces**)**, also adopt a Linda-like coordination model and extend it by defining a reactive tuple space model.

Associative coordination suits mobile-agent applications. In a wide and dynamic environment like the Internet, a complete and updated knowledge of hosting environments and other application agents may be difficult or impossible to achieve. Agents are likely to require pattern-matching mechanisms to deal adaptively with uncertainty, dynamicity, and heterogeneity. Consequently, integrating these mechanisms directly in the information space itself will provide much-needed simplification of agent programming and reduction of application complexity.

In our case study application, associative mechanisms may not be necessary for interagent interactions: As shown previously, agents know exactly which message to retrieve. However, if the application consists of several agent types searching for different keywords, an associative mechanism permits an agent to check the tuple space for the presence of marker messages whose keyword field matches the agent's own keyword. Conversely, the associative mechanism makes the agent ignore those messages left by agents searching for different keywords.

Agent-to-local environment interactions occur as shown in the following example: If the pathnames of all public-readable files reside in the tuple space in the form (name, extension, date, filepointer), agents can simply look for tuples that correspond to pathnames that match the "html" extension, as shown in Figure 6. The actual content of the document can be retrieved by examining the tuple field that refers to the corresponding file.



IN(?, "html", ?, ?)
Tuple space
("Doc", "html", 12/8/98, File@A56)
("Doc", "txt", 12/8/98, File@2E7)
HTML
TXT
(a)
("Doc", "html", 12/8/98, File@A56)
Tuple space
("Doc", "html", 12/8/98, File@A56)
("Doc", "txt", 12/8/98, File@2E7)
HTML
TXT
(b)

Figure 6. The searcher agent (a) asks for a tuple that matches the given template; the tuple space (b) returns a matching tuple with the reference to an HTML file.

ADDING REACTIVITY

The coordination models we've described can be enriched with reactivity. By embodying computational capacity within the coordination media, reactivity lets the application issue specific programmable reactions that can influence the behavior of agent interactions.

In active networks and active messages,[11] reactivity makes it possible for messages to themselves embody the code the system needs to handle them, which implies a move toward dynamic and programmable direct coordination models. In event-based, meeting-oriented coordination models,[7] reactivity lets synchronization objects embody specific policies to influence the interactions among the agents involved in a meeting. For example, synchronization objects can be programmed to asynchronously notify agents of events, thus achieving a partial form of temporal uncoupling, as in Mole.

In a reactive-tuple-space model, the tuple space transcends its role as a mere tuple repository with a built-in and stateless associative mechanism. Instead, tuple spaces can also have their own state and can react with specific actions to the accesses performed by mobile agents. Reactions can access the tuple spaces, change their content, and influence the semantics of the agents' accesses.

In this context, the TuCSoN model[12] defines programmable logic tuple spaces for the coordination of knowledge-oriented mobile agents. Tucson programs reactions as first-order logic tuples. The MARS system implements a portable reactive tuple space model for the coordination of Java-based agents.[5] MARS first defines the tuple space interface according to the JavaSpaces specification, then programs reactions as Java methods associated to tuples.

Tuple space reactivity can provide several advantages in mobile-agent applications. It can be used to implement specific local policies for the interactions between the agents and the hosting environments, to achieve better control, and to defend the integrity of the environments from malicious agents. In addition, reactions can adapt the interactions' semantics to the hosting environments' specific characteristics, making agent programming much simpler than when using the fixed pattern-matching mechanism of Linda-like models.

More generally, being able to adapt the tuple spaces' behavior to specific accesses adds distributed intelligence to the whole system. Given that Internet agents require intelligence and adaptive behavior to function effectively, we believe the same properties can enrich Internet sites. For example, in the same way that an intelligent agent can dynamically evaluate the characteristics of a hosting environment to plan appropriate resource access patterns, a reactive tuple space can dynamically change both its content and the structure of its tuples to adapt it to agent accesses.

Selected URLs for Mobile Agents

The following Web sites contain useful information about various mobile-agent technologies.

Ambit
http://www.luca.demon.co.uk/Ambit/Ambit.html

Ara
http://www.uni-kl.de/AG-Nehmer/index_e.html

D'Agents
http://agent.cs.dartmouth.edu

General Magic
http://www.genmagic.com

IBM Aglets
http://www.trl.ibm.co.jp/aglets

MARS
http://sirio.dsi.unimo.it/MOON

Mole
http://mole.informatik.uni-stuttgart.de

PageSpace
http://flp.cs.tu-berlin.de/pagespc

TuCSoN
http://www-lia.deis.unibo.it/Research/TuCSoN

In our case study application, we can exploit a reactive tuple space in several ways, even if we take into account only simple reactions that lack peculiar intelligent behavior. Suppose that one hosting environment represents a local Web server whose HTML pages are replicated at one or several mirror sites. In this case, not only must the other searcher agents of the same application avoid multiple visits to a site, but they should also avoid visiting the mirror sites. The local administrator can program a reactive tuple space to react to the local insertion of marker messages by coordinating itself with mirror sites and replicating the agents' marker messages there. The site administrator could also decide to exploit the reactivity to delete, on an allowed-lifetime basis, the marker messages left locally by searcher agents. This provision could be useful because of the intrinsic difficulty searcher agents have in knowing if a visited site's marker message will be needed by other searcher agents.

As a further example, recall that the Linda model defines a disruptive "in" operation that extracts matching tuples from the space. Suppose a malicious attempt to extract HTML tuples from a site occurs. The local administrator could decide not to raise any exception, but to program its local tuple space to transform this disruptive, foreign-agent operation into a nondisruptive read operation. Doing so lets the malicious agent proceed without alerting it to the innocuous effect of its access, while the administrator preserves the local environment's safety.

Further advantages could be provided by installing reactions in the tuple spaces of the visited sites that let agents decide their own coordination rules. For example, in our case study application, a searcher agent could install a reaction in the visited tuple space. This reaction prohibits further incoming searcher agents from accessing those HTML files whose corresponding pages have not been modified since the last visit.

In general, an application's ability to install its own reactions into tuple spaces permits a clear separation of concerns between algorithmic and coordination issues:[3] The agents embody the problem-solving algorithms; the reactions represent the application's specific coordination rules. This arrangement simplifies both application design and agent coding by allowing the programmer to separate algorithmic and coordination issues. For example, the code for a basic search algorithm may reside in the agents, while the code to avoid multiple visits resides in the tuple space reactions.

The choice of coordination model greatly affects the design of mobile-agent applications. In particular, a Linda-like coordination model—possibly enriched with programmable reactive capabilities—can lead to a clean, flexible, and scalable application design. We do not claim that reactive tuple spaces must be the exclusive model for every kind of application, however. Performance reasons may, for example, suggest using direct communication among distant agents instead of forcing agents to move to the same site to interact via a tuple space. In this context, a quantitative analysis—constructed via simulation tools or in the field by measuring the performance of different design solutions —would help designers identify these situations and select the most suitable approach.

Additional challenges must be overcome before both mobile agents and coordination technologies can gain wide acceptance. In general, mobile-agent infrastructures should integrate appropriate tools to deal with problems such as distributed agents' termination and garbage collection, agents' recovery protocols, and protection of the agents' internal information from malicious hosts. In particular, a coordination infrastructure must address the following issues:

- Effective security policies must be defined to rule agent interactions. In the case of reactive tuple spaces, the spread of application-specific reactions over foreign Internet sites must be ruled, too.

- Letting mobile agents access a node and its local resources implies resource consumption and, therefore, should be properly monitored and regulated, possibly via the definition of coordination models enriched with contracting and currency-exchange capabilities.
- Any coordination proposal must be well integrated with the current Web infrastructure, including CORBA applications and Web services and browsers. Such a proposal must also be made compliant with existing standards, such as HTTP and XML, to facilitate interoperability with existing components.

These issues represent important areas for study and opportunities for new product development. ❖

Acknowledgments

We thank the anonymous referees for their helpful suggestions. This work has been supported by the Italian Ministero dell'Università e della Ricerca Scientifica, within the framework of the MOSAICO—Design Methodologies and Tools of High-Performance Systems for Distributed Applications—project.

References

1. N.M. Karnik and A.R. Tripathi, "Design Issues in Mobile-Agent Programming Systems," *IEEE Concurrency*, July-Sept. 1998, pp. 52-61.
2. A. Fuggetta, G. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Trans. Software Eng.*, May 1998, pp. 342-361.
3. D. Gelernter and N. Carriero, "Coordination Languages and Their Significance," *Comm. ACM*, Feb. 1992, pp. 96-107.
4. G.A. Papadopoulos and F. Arbab, "Coordination Models and Languages," *Advances in Computers*, Academic Press, Orlando, Fla., Aug. 1998, pp. 329-400.
5. G. Cabri, L. Leonardi, and F. Zambonelli, "Reactive Tuple Spaces for Mobile Agent Coordination," *Proc. 2nd Int'l Workshop Mobile Agents*, *Lecture Notes in Computer Science*, No. 1,477, Springer-Verlag, Stuttgart, Germany, 1998, pp. 237-248.
6. H. Peine, "Ara—Agents for Remote Action," *Mobile Agents: Explanations and Examples*, W.R. Cockayne and M. Zyda, eds., Manning/Prentice Hall, Upper Saddle River, N.J., 1997, pp. 96-161.
7. J. Baumann et al., "Mole—Concepts of a Mobile Agent System," *World Wide Web J.*, Vol. 1, No. 3, 1998, pp. 123-137.
8. L. Cardelli and D. Gordon, "Mobile Ambients," *Foundations of Software Science and Computational Structures, Lecture Notes in Computer Science*, No. 1,378, Springer-Verlag, Stuttgart, Germany, 1998, pp. 140-155.
9. P. Domel, A. Lingnau, and O. Drobnik, "Mobile Agent Interaction in Heterogeneous Environments," *Proc. 1st Int'l Workshop Mobile Agents*, *Lecture Notes in Computer Science*, No. 1219, Springer Verlag, Stuttgart, Germany, Apr. 1997, pp. 136-148.
10. P. Ciancarini et al., "Coordinating Multi-Agent Applications on the WWW: A Reference Architecture," *IEEE Trans. Software Eng.*, May 1998, pp. 362-375.
11. D. Tennenhouse et al., "A Survey of Active Network Research," *IEEE Communications*, Jan. 1997, pp. 80-86.
12. A. Omicini and F. Zambonelli, "TuCSoN: A Coordination Model for Mobile Agents," *J. Internet Research*, Vol. 8, No. 5, 1998, pp. 400-413.

Giacomo Cabri is a PhD student in computer science at the University of Modena and Reggio Emilia. His research interests include tools and environments for parallel and distributed programming, wide-scale network applications, and object-oriented programming. He has a Laurea degree in electronic engineering from the University of Bologna. He is a member of the Italian Association for Object-Oriented Technologies (TABOO).

Letizia Leonardi is an associate professor in the Department of Engineering Science at the University of Modena and Reggio Emilia, where she teaches basic and advanced computer science courses. Her research interests include object-oriented programming environments; parallelism and distribution issues, especially as they apply to object systems; and design and implementation of parallel object environments in distributed, massively parallel, and heterogeneous architectures. Leonardi has a Laurea degree in electronic engineering and a PhD in computer science from the University of Bologna. She is vice-president of TABOO and a member of AICA.

Franco Zambonelli is a research associate in computer science at the University of Modena and Reggio Emilia. His research interests include parallel, distributed, and Internet programming; distributed algorithms for fault-tolerance; and debugging. Zambonelli received a Laurea degree in electronic engineering and a PhD in computer science from the University of Bologna. He is a member of the IEEE, ACM, EuroMicro, and TABOO.

Contact Cabri, Leonardi, and Zambonelli at {giacomo.cabri, letizia.leonardi, franco.zambonelli}@ unimo.it.