

Engineering Mobile-agent Applications via Context-dependent 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, mobility introduces peculiar coordination problems in agent-based 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 an infrastructure enforcing context-dependent coordination can be effectively exploited to simplify and make more modular the design of Internet applications based on mobile agents. In addition, the MARS coordination infrastructure is presented as an example of a system in which the concept of context-dependent coordination has found a clean and efficient implementation.

Keywords:

Internet Applications, Mobile Agents, Coordination

1 INTRODUCTION

Internet Agents

Designing and deploying Internet applications introduces new problems and requirements over traditional approaches to distributed program development. On the one hand, *decentralization* of management, *wideness* of the distribution scale, and the necessary *partial knowledge* available about the target environment, require application components to integrate the capacity of dealing with unexpected situations as part of their intrinsic behavior, rather than in terms of “exceptions”. On the other hand, the intrinsic *interactivity* of Internet applications require application components to interact with a variety of other, possibly unknown in advance, entities (such as services, legacy applications, other

components of different applications). These kinds of interactions often have to assume complex forms, difficult to be expressed in terms of a traditional client-server fashion, and leading to an “organizational” view of the distributed application rather than to a “compositional” view, as in the case of distributed object-oriented applications.

All of the above reasons justify the adoption of an agent-oriented approach for Internet application development [Zam00]. By conceiving and developing applications in terms of autonomous entities, capable both of reacting to changes in the environment and of executing in a proactive way, one can more naturally deal with dynamicity and unpredictability [Jen00]. 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 [DemC96].

Agent Mobility

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* (e.g., Internet nodes or administrative domains) [CarG98, FugPV98] and to design applications in terms of agents that are aware of the distributed nature of the target and *logically move* (when not even physically) in different execution environment to access to the resources there allocated.

Coordination in the Presence of Mobility

Agents, during their execution, are in need to interact, communicate, and synchronize their activities, both with other agents (inter-agent coordination) and with resources on the visited execution environments (agent-environment

coordination).

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 interaction spaces (e.g., blackboards or tuple spaces), each associated to an execution environment. These interaction spaces are in charge of mediating all coordination activities (both inter-agent and agent-environment ones) for the agent currently executing (whether logically or physically) in the associated execution environment [OmiZ99, CabLZ00a]. On the one hand, such an infrastructure naturally matches the network-aware – logically mobile – nature of Internet agents and enforces the principle of locality in interactions. On the other hand when agents can physically move at a world-wide scale, relying only on global inter-agent interactions is not suitable, requiring the introduction of complex communication middleware and/or forcing odd design choices [CabLZ00a].

What is Context-Dependency?

For the very fact that agents move onto different execution environments during their execution, and will exploit the associated local interaction spaces, they will access to different data, services, and will interact with different agents, depending on the environment. In other words, the same interaction event is likely to produce different result (and to lead to different actions on the performing agent) depending on the interaction space via which it occurs. This form of context-dependency is intrinsic in agent mobility, and the interaction space does not play any active role in it. However, more sophisticated forms of context-dependent interactions may be needed in Internet applications implying an active role of the interaction space:

- each execution environment has its own specific characteristics and security policies, and it may be somehow in need of enacting specific coordination laws on the agents accessing its local interaction space (*environment-dependent coordination*);
- application agents may require their coordination activities to occur according to specific application needs, despite the different characteristics of the local environment and the different coordination laws there enacted (*application-dependent coordination*).

By enhancing interaction spaces with the capability of dynamically programming their behaviors in response to interaction events [DenNO98, CabLZ00a], one can make interaction spaces behave in different ways depending on which agent issued which interaction event. This characteristic can be used to support a strong form of *context-dependent coordination*. In particular: (i) the administrator of one site can adapt the behavior of the interaction spaces to enforce – transparently to agents – to enforce site-specific policies/laws on the local interactions; (ii) application agents can dynamically adapt the behavior of

the visited interaction spaces accordingly to specific application needs.

Contribution and Structure of the Paper

The aim of this paper is to introduce and analyze the concept of context-dependent coordination, showing its possible exploitation in the design and development of Internet-agent applications.

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 in the design and development of Internet applications. In particular, this section shows how context-dependent coordination promotes a clear separation of concerns between computational and coordination issues, which is likely to make applications more modular and easy to be maintained. Section 4 presents shows how context-dependent coordination can find an effective implementation in MARS, a coordination infrastructure based on programmable tuple spaces and firstly described in [CabLZ98]. Section 5 discusses related work in the area. Section 6 concludes the paper, outlining open research directions.

2 FROM GLOBAL AND COUPLED TO LOCAL AND UNCOUPLED INTERACTIONS

Agent mobility discourages the generalized adoption of traditional coordination models, such as peer-to-peer message-passing, client-server, globally distributed shared memories, relying on the presence of a global communication abstraction. First, the globality of the Internet scenario and, therefore, of the space of possible interactions, can make communications between distant, physically mobile, 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) usually enforced by these models. Third, global and coupled interactions clashes with that principle of locality in interactions that have always to be enforced in the design and development of complex software systems and that, in the case of Internet agents, calls for geographical locality in interactions.

The suitable 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, global, interaction between them is not feasible without the availability of complex communication middleware to cope with physical agent mobility: searcher agents are forced to refer to a fixed, well-known, “home agent”, acting as mediator (or as a centralized shared memory) for all searcher agents. This introduces performance and reliability problems, and also complicates the application design. Instead, a local infrastructure based on tuple spaces can be exploited in a very simple and elegant way, leading to a simpler 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. Of course, the tuple space can be effectively used also to access the information about the local HTML files. The administrator can put in the tuple space tuples such: (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 CONTEXT-DEPENDENT COORDINATION

A model of local and uncoupled interactions not only fits both the physical and the logical mobility of agents, but also invites thinking applications in terms of mobility. When an agent migrates to a new execution environment, its interaction space changes accordingly. Thus, even if the interaction model (e.g. tuple-based) provided by the local interaction spaces is the same independently of the environment, the agent's perceivable world changes as a consequence of its movement: what it obtains from a given interaction event on a site is likely to be different from what it obtains on a different site.

The above form of context-dependent coordination is intrinsic in agent mobility and in the adoption of independent, local interaction spaces. However, if the interaction space is simply an infrastructure for storing and forwarding of data, messages, or tuples, the above simple form of context-dependency is likely to be very complicated to be handled by agents. First, coordinating with the other agents residing in a foreign environment that an agent is likely to meet 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 the case of a static interaction space the above problems are likely to lead to huge agents, requiring costly maintenance activities as soon as their coordination requirement changes and/or the external conditions (i.e., the execution environments) changes. 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 [AhuCG86] towards a *programmable tuple space model*: for any kind of access events (i.e., request for storing, reading, or extracting a tuple), a new behavior in response to it can be programmed to override the basic behavior of the tuple space (i.e., storing and retrieval of tuples based on a stateless 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, leads to a stronger, and highly-manageable, form of *context-dependent coordination*. 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 coordination laws* 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 laws* 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.

The adoption of programmable interaction spaces to enforce context-dependent coordination leads to the scenario depicted in Figure 1. On the one hand, agents on a site, whatever the application they belong to, are subject to the local environment-dependent laws, which can be different from environment to environment. On the other hand, the agents of a given application can spread their own coordination laws on the sites they visit, and these coordination laws will influence the coordination activities of all the agents belonging to the same application.

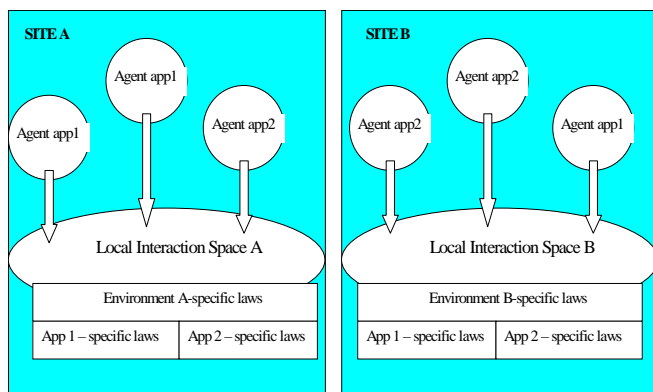


Figure 1. The Scenario of Context-Dependent Coordination

In the following of this section, we describe several cases for which programmable interaction spaces (specifically, programmable tuple spaces) can be effectively exploited in Internet-agent applications to lead to both environment-dependent and application-dependent coordination. Finally,

we discuss how the possibility of clearly separating the definition of the agents internal activities from the definition of the environment-dependent and of the application-dependent coordination laws can positively impact in the design and development of Internet-agent applications.

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.

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". In other words, agents have to explicitly look for this information into the visited tuple spaces. This is likely to complicate the agent design and requires an agent to know *a priori* that it has to look for some information. In the presence of a programmable tuple space model, a simpler and more general solution can be sketched: agents can install in the visited tuple spaces a reaction that make the tuple spaces reflects – in term of the produced information – the knowledge/information acquired, to be implicitly transferred to other application agents. With regard to the above example, an agent that has visited a site can install a reaction that returns, to other agents of the same application requesting HTML files, only those files that have been modified since the 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. This example shows that 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, to free agents from the duty of actually controlling the execution of the protocol.

Impact on Design and Development

Given the availability of an infrastructure based on programmable local interaction spaces, the adoption of a context-dependent coordination model can have a very positive impact on the engineering of mobile agent applications. In fact, the model naturally invites in designing an application by clearly separating the intra-agent aspects – related to the specific computational roles of the application agents – and the inter-agent ones – related to the interaction of the agents with the other agents of the same application and with the visited execution environments. In other words, the model promotes a clear separation of concerns, which is likely to reduce the complexity of application design.

In the case study example, the application designer can independently focus on two different classes of design issues:

- *intra-agent engineering*: what kind of information my application agents should retrieve on the visited sites? How should they analyze this information and extract useful data from it? How should this data reported back to the user? How should my agents select their itinerary and possibly clone themselves?
- *inter-agent engineering*: how should my agents influence with each other when they visit the same site? What kind of information should they exchange via the programmable interaction space?

The analysis of the inter-agent engineering issues leads to the definition of the application-dependent coordination laws to be spread into the visited sites by application agents, which are independent from the design choices deriving from the analysis of the intra-agent engineering issues. For instance, no matter what the method for information extraction integrated into the application agents, the designer can opt either for having agents retrieve no HTML pages at all from a site that has already been visited in the past, or from having agents retrieve only those HTML pages that have been modified since the last visit.

Independent from the role of the application designers and developers is the role of site administrators. When new kinds of application agents are going to be deployed on the Internet, the administrators of one site can design and implement all the environment-dependent coordination laws that they may find it necessary to both facilitate the execution of the agents on their site and protect their site from improper exploitation of the local interaction space.

The separation of concerns of the design phase is preserved during the development and maintenance phases too: the code of the agent is separated from the code implementing the coordination laws (whether environment- or application-dependent), and they can be coded, changed, and re-used, independently.

In the case study example, if the application developers want to add some more “intelligence” to the agents – e.g., by having them integrate more sophisticated forms of analysis and extraction of data from HTML pages – they can change the code of the agents without influencing at all the code implementing the application-specific coordination laws. Conversely, if they want to change the application-specific coordination laws they can do that without having to change the code of the agent. As an example, let us suppose that the application-specific coordination laws for the case study make an agent arriving on a site retrieve only those HTML pages that have been modified since the last visit. However, later in time, the designer and/or the developers can discover that, to achieve effective information analysis, an agent arriving on a site does not need to analyze only the HTML pages that have been modified since the last visit, but also all those unmodified pages that are linked to the modified ones. Since the coordination laws are clearly separated from the code of the agents, they can be easily integrated in the application by having the unchanged agents distribute over the visited site the updated coordination laws. A similar approach can be followed by the administrator of one site, that can change at any time its environment-specific laws without forcing the agents that wants to visit the site to adapt to the changes.

4 CONTEXT-DEPENDENT COORDINATION IN MARS

MARS (Mobile Agent Reactive Spaces), developed at the University of Modena and Reggio Emilia, and described in [CabLZ98, CabLZ00b], implements a tuple-based coordination architecture for Java mobile agents, which is portable to different mobile agent systems. The characteristics of MARS and of its programmable tuple space model make it a suitable infrastructure for context-dependent coordination model.

The MARS Architecture

Globally, the MARS architecture is made up of a multiplicity of independent programmable tuple spaces, each one associated to a node and accessed by the agents locally executing in that node (see Figure 2). When an agent arrives on a site, it has to be automatically bound to the interaction space of the local environment, i.e., the local MARS tuple space. Therefore, the only requirement to integrate MARS with a mobile agent system is to make the *agent server* – in charge of accepting incoming agents – supply agents with a reference to the local MARS tuple space, which can then be accessed by agents via Linda-like primitives.

To enforce a context-dependent coordination model, the

MARS tuple space on a node should be intended as the only resource an agent can access on that node, and it should be therefore exploited for both inter-agent and agent-environment coordination. However, the current MARS architecture is conceived as an additional coordination service for agents, and does not rule out the possibility for agents of exploiting any other form of coordination supported by the agent system.

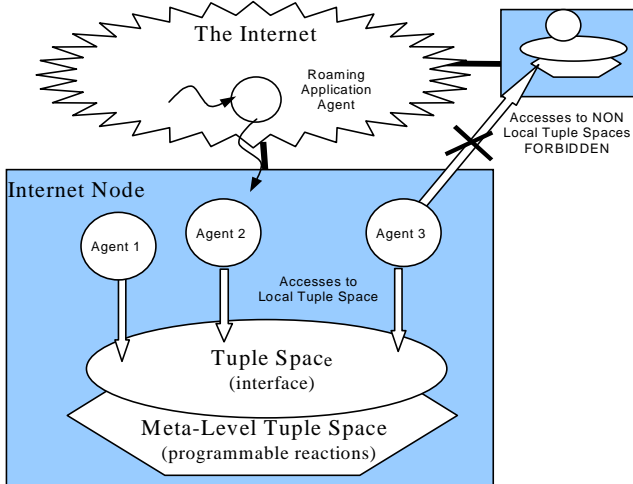


Figure 2. The MARS Architecture

The MARS Interface

MARS has been designed in compliance with the Sun's *JavaSpaces* specification. Therefore, as in *JavaSpaces*, MARS tuples (also called “Entries”) are Java objects whose instance variables represent the tuple fields. Any field of a tuple can have either a defined (actual) or a null (formal) value. Each MARS tuple space, in its turn, is a Java object, instance of a class implementing the MARS interface and extending the *JavaSpace* one. To access the tuple space, the following operations are provided:

- write, to put a tuple, supplied as the first parameter, in the space;
- read/take, to read/extract a tuple from the space, on the basis of a template tuple supplied as the first parameter and to be used as a pattern for the matching mechanism;
- readAll/takeAll, to read/extract all matching tuples from the space.

The template tuple supplied to the input operations can have both defined and null values. A tuple *T* matches a template tuple *U* if the defined values of *U* are equal to the corresponding ones in *T*. The matching rules take into account the serialized object form and provide for subclass-superclass matching.

With reference to the application example, the tuples representing HTML files and the corresponding templates can be implemented by the Java class shown in Figure 3.

The figure also show the code fragment of an agent creating an object from this class to define a template and, on a site, use this template to retrieve, via a non-blocking `readAll` primitive, all the tuples representing HTML files and matching with it.

```
class FileTuple extends AbstractEntry
{ // tuple fields
  public String FileName;      public String Extension;
  public String Keyword;
  public Date ModificationTime; public int Size;
// constructor of the tuple.
public FileTuple(String name, String extension,
  String keyword, Date modificationTime, int size)
{  FileName = name; Extension = extension;
  Keyword = keyword;
  ModificationTime = modificationTime; Size = size;
}
}
...
//code fragment for the searcher agent
// creation of a template
FileTuple filePattern =
  new FileTuple(null, "html", "coordination", null, null);
...
// access the local space to retrieve the tuples representing
// HTML files related to the specific keyword
Vector HTMLFiles =
  LocalMARSSpace.readAll(filePattern, null, NO_WAIT);
...
```

Figure 3. An agent that defines a template and accesses the tuple space

The MARS Reactive Model

The association of reactions to access events is represented via *4-plets* stored in a “meta-level” tuple space. A meta-level tuple has the form of (Rct, T, Op, I) : it means that the reaction method (representing the reaction itself) of the object *Rct* will be triggered when an agent with identity *I* invokes the operation *Op* on a tuple/template matching *T*. Putting and extracting tuples from the meta-level tuple space means installing and uninstalling, respectively, reactions associated to events at the base-level tuple space. A meta-level 4-ple can have some non-defined values, in which case it associates the specified reaction to all the access events that match it. For example, the 4-ple $(ReactionObj, null, read, null)$ in the meta-level tuple space associates the reaction of the *ReactionObj* instance to all read operations, whatever the tuple type and content and the agent identity. When an access event to the base-level tuple space occurs, MARS issues the pattern-matching mechanism in the meta-level tuple space to look for reactions to be executed in response to the access event. If several 4-plets satisfy the meta-level matching mechanism, all the corresponding reactions are executed in a pipeline, accordingly to their installation order. 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), as exemplified in the following.

```
class HTML2HTML implements Reactivity
{ public Entry[] reaction(Space s, Entry FT,
                        Operation Op, Identity Id)

// no match already occurred if the site has only htm files
{ // modifies the extension of the required files
  FT.Extension = "htm";
  // require matching with new extension
  Entry [] result = s.readAll(FT, null, 0);

  for (int i=0; i < result.length; i++)
  // change back the extensions in the found tuples
    result[i].Extension = "html";
  return result; }
}
```

Figure 4. A reaction that makes HTML match with HTML

Environment –Dependent Coordination in MARS

As described in Subsection 3.1, if a site stores its HTML files with the “htm” extension instead of the “html” ones, the administrator can exploit the MARS programmability to enable agents to retrieve anyway information about its HTML files, without forcing them to deal with this heterogeneity and without having the administrator forced to change local filenames or duplicate the FileTuple tuples in both “html” and “htm” forms. The administrator can install the reaction shown in Figure 4, which transforms any request for file tuples with “html” extension into a request for tuples having “htm” as extension field. The reaction can be installed by writing the meta-level tuple: (HTML2HTML_instance,(null,"html",null,null,null),readAll,null) in the meta-level tuple space, to express that the reaction in the object HTML2HTML_instance has to be triggered when any agent issues a readAll operation by providing the template template (null,"html",null,null, null).

As a second example, when an agent tries to perform an unauthorized takeAll operation on the “html” tuples, the system administrator can decide not to raise any exception, and to let the agent read the content of the matching tuples without deleting them. To this purpose, (s)he can install a reaction that transparently transforms any disruptive takeAll operation, performed by a foreign agent on the FileTuple tuples, into a non-disruptive readAll operation. This reaction is implemented by the ReadOnly class in figure 5 and can be installed by writing the 4-ple (ReadOnly_instance, (null, “html”, null, null, null), takeAll, null) in the meta-level tuple space.

The behavior of the two above reactions can be combined in

a harmless way: the tuple space can be made react with an HTML2HTML-ReadOnly pipeline to takeAll operations requesting for “html” tuples, and have the ReadOnly reaction executed for each of the “htm” tuples returned by the HTML2HTML reaction. Also, the reactions can be designed independently of each other and independently of applications agents, and can be changed/update at any time, without affecting the code of the application agents.

```
class ReadOnly implements Reactivity
{public Entry reaction(Space s, Entry FT,
                    Operation Op, Identity Id)
{ if(matched(FT)) { // if a match has been produced
  // check for the identify of the agent
  //performing the operation
  if (Id.equals(manager))
    // the administrator can delete the tuple
    return s.take(FT, null, 0);
  // otherwise, the tuple is returned but it not extracted
  else return FT }
  else return null;} // no match has been produced
}}
```

Figure 5. A reaction that transforms a disruptive operation into a non-disruptive one

```
class ModifiedOnly implements Reactivity
{ private Date visit; // date of the last visit
  public ModifiedOnly() // constructor
  { visit = new Date(); }
  // date of last visit automatically set by the constructor

  public Entry reaction(Space s, Entry FT,
                    Operation Op, Identity Id)
  { // is this a matching document having been
    // modified after last visit?
    if(s.matched(FT) &&
      !(FileTuple)FT.ModificationTime.before(visit))
      { // set the time of the last visit
        visit = new Date();
        // return the updated tuple
        return FT; }
    // if no modification, no tuple has to be returned
    else return null; }
}
```

Figure 6. A reaction that returns only modified HTML files

Application-Dependent Coordination in MARS

As described in Section 3.2, agents can avoid duplicated work on the same site by retrieving information only about the files that have been modified since the last visit. To this end, any agent that visits a site can install a stateful reaction that stores the information about the time of the visit. Then, if further agents of the same application visit again the site and try to access the FileTuple tuples, the reaction provides for returning to it only the tuples related to the files which have been modified. This reaction, implemented by the ModifiedOnly reaction class shown in figure 6, can be installed by making agents write the 4-ple (ModifiedOnly_instance, (null, “html”, null, null), readAll,

specific_app_id) in the meta-level tuple space, where specific_app_id is the local identifier for the agent of that application, making the reaction affect only that application.

Also in this case, the application-specific reaction can be pipelined in a harmless way with the previously described reactions and it can be independently designed and changed without affecting other application components.

The MARS Security Model

MARS assumes that the mobile agent system – like most of the available systems – provides for agent identification and authentication before allowing any agent to execute on one node. On this basis, the main security mechanism provided by MARS is the association of *Access Control Lists (ACLs)* to tuple spaces and enclosed tuples. Such ACLs define *who* can do *what* on tuple spaces and on specific tuples, and the system administrator can decide whether a given operation performed by a given agent on a given tuple is authorized or not. The access control policy based on ACL can then be complemented with a specific security-oriented programming of the tuple space, as described in Subsection 3.1.

A similar ACL-based security policy is applied with regard to the meta-level tuple space. This can be used to allow/deny agents to install and uninstall reactions in the meta-level tuple space. At a finer level, one can control which set of access events an agent can influence via reactions. In general, the administrator of one site should define proper role authorization policies for external application agents, so as to guarantee that an external agent cannot influence – by installing/uninstalling reactions – the activities of other agents but those of its specific application. In other words, an agent must not be allowed to influence those coordination activities occurring in a different application context.

To this end, if an unauthorized agent installs a reaction that would potentially apply to agents external to its specific application, the security mechanisms guarantee that the reaction indeed apply only to that specific application agents. For example, if the ModifiedOnly reaction of Figure 6 is installed by an agent writing in the meta-level tuple space the 4-tuple (ModifiedOnly_instance, (null, "html", null, null), readAll, null) (in which the last null field specifies the reaction has to apply to all agents) MARS actually writes the tuple (ModifiedOnly_instance, (null, "html", null, null), readAll, specific_app_id), in which the last specific_app_id field is the local identifier of the agent of that application.

5 RELATED WORK

A few of the proposals in the area of Internet 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.

The *T Spaces* project at IBM [IBM98] defines Linda-like tuple spaces to be used as general-purpose information

repositories. However, rather than aiming at defining agent-oriented coordination media, T Spaces aims at providing a powerful and standard interface for accessing large amounts of information organized as a database. For this reason, T Spaces recognizes the limits of the basic Linda model and integrates a peculiar form of programmability, by enabling new complex primitives (i.e., queries) to be added to a tuple space. This approach makes T Spaces less usable in the open Internet environment, since it requires application agents either to be aware of the operations available in a given tuple space or to somehow dynamically acquire this knowledge. Also, T Spaces does not define architecture conceived to meet the problem of mobility: agents can refer to multiple tuple spaces, whether local or remote, and access them in a location-unaware way.

The *TuCSon* model [OmiZ99], developed in the context of an affiliated research project, adopts an architectural model very similar to that of MARS, 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. However, in our opinion, this is likely to increase the complexity of the agent, which has to explicitly handle the URLs references to remote tuple space. With regard to the tuple space model, TuCSon resembles MARS in its full programming capability of tuple spaces, but 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 faces 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. Thus, 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 on which are the specified rules for tuple flowing across tuple spaces.

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 laws. An agent can initiate a group, and fix the laws to be respected by other agents willing to enter and interact within the group. These laws are typically security-

oriented, and express which operations the agents are allowed to perform, and in which order. To enforce these laws, 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 laws. 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 issues, it has the full potential for enforcing context-dependent coordination in complex systems of non-mobile agents interacting in a peer-to-peer way.

6 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 implemented by exploiting the notion of programmable tuple spaces. 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, Jen00], due to their importance in the analysis and design of multi-agent applications, have to be somehow integrated in the model; *(iii)* more 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.

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.
- [CabLZ00a] G. Cabri, L. Leonardi, F. Zambonelli, "Mobile-Agent Coordination Models for Internet Applications", IEEE Computer, Vol. 33, No. 2, pp. 82-89, Feb. 2000.
- [CabLZ00b] G. Cabri, L. Leonardi, F. Zambonelli, "MARS: a Programmable Coordination Architecture for Mobile Agents", IEEE Internet Computing, Vol. 4, No. 4, July-Aug. 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 et al., "KQML as an Agent Communication Language", 3rd International Conference on Information Knowledge and Management", November 1994.
- [FugPV98] A. Fuggetta, G. Picco, G. Vigna, "Understanding Code Mobility", IEEE Transactions on Software Engineering, Vol. 24, No. 5, pp. 352-361, May 1998.
- [IBM98] "T Spaces: the Next Wave", IBM System Journal, Vol. 37, No. 3, pp. 454-474, 1998.
- [Jen00] N. R. Jennings, "On Agent-Based Software Engineering", Artificial Intelligence, Vol. 117, No. 2, pp. 277-296, 2000.
- [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. Minsky, V. Ungureanu, "Law-Governed Interaction: A Coordination & Control Mechanism for Heterogeneous Distributed Systems", ACM Transactions of Software Engineering and Methodology, 2000, to appear.

- [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 *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*, Springer, 2000.