Coordination Technologies for Internet Agents

Paolo Ciancarini
DSI - Università di Bologna
Mura Anteo Zamboni, 7 Bologna, Italy
`ciancarini@cs.unibo.it`

Andrea Omicini
LIA - DEIS - Università di Bologna
Viale Risorgimento, 2 Bologna, Italy
`aomicini@deis.unibo.it`

Franco Zambonelli
DSI - Università di Modena
Via Campi, 213b Modena, Italy
`franco.zambonelli@unimo.it`

Abstract

Since its birth the Internet has always been characterised by a twofold aspect of a distributed information repository, to store, publish, and retrieve program and data files, and an interaction medium, including a variety of communication services. The current trend is to merge the two aspects described above, and envision the Internet as a globally distributed computing platform where communication and computation can be freely intertwined. However, traditional distributed programming models fall short in this context, due to the peculiar characteristics of the Internet. On the one hand, Internet services are decentralised and unreliable. On the other hand, even more important, mobility, either of users, devices, or application components, is going to impact the Internet in the near future. Since Internet applications are intrinsically interactive and collaborative, the definition and development of an appropriate coordination model, and its integration in forthcoming Internet programming languages, is a key issue to build applications including mobile entities. We sketch the main features that such a model should present. Then, we survey and discuss some coordination models for Internet programming languages, eventually outlining open issues and promising research directions.

1 Introduction

Traditional computing models and the corresponding programming languages are evolving in order to match the novel requirements of a networked world offering, via Internet and the Web, global communication and information services. The increasing need of networked applications requires the development of geographically distributed software systems which can exploit the large availability of hardware resources at a low cost. However, the growth of wider and wider networks introduces new issues to face, such as mobile computing [8].

New computing paradigms based on code mobility are not only intrinsically suited to a world-wide and mobile computing scenario, but they also provide advantages in terms of efficiency and reliability [16]. However, they introduce the need for specification methods and languages able to deal with the specific features of architectures supporting mobility of code [27, 19, 5]. For instance, since the original WWW architecture supports very limited forms of distributed programming, it has been extended with network-aware programming languages, like Java, which extends the functionality of WWW browsers in order to support *applets*, that are made of code loaded on demand from a remote site. Some Java-based applications are even based on migratory agents [21], namely processes which can decide to move to another site in order to look for some resources, and which need to be controlled in their travelling over the network.

Despite the great deal of activity in this areas, several issues have to be faced to make mobility used and usable. A key issue when a software system includes mobile components is the continuous reconfiguration of its software architecture. Currently there are a few formal notations able to deal with dynamic structural changes. A paper which analyses a number of formal models suitable for describing architectures including mobile components is [15]. Another important issue in all models is how mobile entities interact with their environment [4] and, in turn, how the environment can control a mobile entity.

These issues are complex also because when dealing with mobility there is some confusion among three different network concepts, which should be kept distinct:

- The *physical* network, made mostly of immobile hosts and reliable and fast connections, where a mobile entity consists of a piece of hardware using a wireless connection. The main problem mobility poses at this level is that wireless connections are usually unstable and offer low bandwidth.

- The *middleware network*, made mostly of abstract machines and software services, where a mobile entity consists of a process closure migrating from a host to another host. Some problems mobility poses at this level are dynamic re-configuration (eg. dynamic binding of processes to execution environments) and service reliability (eg. security and quality of service).

- The *logical network*, made of application code scattered over the middleware network, where a mobile entity consists of some executable code migrating from an abstract machine to another one. An interesting problem at this level is how such a mobile entity can cooperate with other entities in the application.

In this paper we are interested in the last form of mobility, namely over a logical network, where the migrating entity is usually called a *software agent*. In particular this paper aims to survey and analyse some coordination models for languages for programming applications including mobile software agents.

The paper is organised as follows. Section 2 discusses two main issues influencing the design of an Internet programming language, namely first class mobile entities, and a coordination model to control them. Section 3 describes what is a coordination model suitable to mobile entities, showing that a coordination model based on *tuple spaces* exhibits some important features. Section 4 focuses on the current Web architecture and surveys some architectural solutions to integrate in the WWW a tuple space coordination model. Some language design choices are analysed with respect to the intended coordination architecture and its impact on agent programming. Section 5 sketches an ideal coordination architecture for Web-based interacting mobile entities. Finally, Section 6 outlines some open issues and related promising research directions.

2 Mobility and Interaction in Internet Applications

The Internet can be no longer considered as a collection of network services, or as a mere information repository. The continuous advances in network technologies, along with the widespread diffusion of the WWW, suggest to see the Internet as a new, *globally distributed computing platform*, calling for new programming paradigms, models, and languages.

In this section we point out those features of the current Internet architecture which make traditional distributed programming approaches inadequate for the development of Internet-based applications. In particular, we argue that the definition of a suitable coordination model is a key issue to any programming model and language for Internet application development.

2.1 Structuring an Internet Application

The Internet, even if considered just as a world-wide "network of networks", exhibits some features which differentiate it from the traditional infrastructure for parallel/distributed applications. In particular, the Internet is not under the management of a single organisation. Consequently, Internet nodes are heterogeneous not only in terms of hardware but also in terms of software configuration. The Internet is also intrinsically dynamic, as new nodes continuously appear and old nodes either change their configuration or simply disappear. Partially as a consequence of the above points, and also because of its wide size, the Internet infrastructure is intrinsically unpredictable (eg. sites can join and leave the network) and unreliable (eg. the network can lose messages).

The above reasons justify the current quest for new programming models and languages for designing and developing Internet applications. In fact, conventional programming models for "traditional" distributed systems either neglect the issues of heterogeneity, dynamicity, and reliability, or they deal with these issues as they were unrelated to the application level.

For the purposes of this paper we model an Internet application using a three-layers architecture:

- the *physical layer*, made up of computing and network devices;
- the *middleware layer*, generally made up of abstract machines, such as traditional operating systems, network services, language executors (eg. Java interpreters);
- the *application layer*, made up of application code scattered over the middleware layer.

The physical layer is composed of heterogeneous computing resources, let them be PCs or workstations, and network devices, such as bridges and routers. However, some recent technological developments will soon make the Internet be populated of widely different devices. On the one hand, network devices are becoming more and more "intelligent", ie. autonomous and adaptable, and enclose richer computing capabilities. On the other hand, there is a great deal of research efforts to enable Internet connectivity for mobile devices, such as laptop and personal digital assistants. In particular, *mobility* is the feature that is likely to have more impact on language design. In fact, applications must take into account the fact that components may execute not only in the context of traditional distributed systems connected to the network, but also in the context of temporarily (or usually) disconnected systems.

The middleware layer, traditionally made up of UNIX IP services, such as `finger`, `sendmail`, and `telnet`, is currently being enriched with the possibility of exploiting a larger number of services and operating environments. Apart from pervasive WWW services, legacy application can be made available and provide services to the world, either via CGI or CORBA interfaces. In the near future, we expect also that most Internet sites will be willing to accept the execution of mobile components, such as simple fragments of code (as in current prototype active routers), procedure code (as in the servlets and applets approach), autonomous agents and whole application closures (as in the case of Jumping Beans [1]).

All these issues carry interesting problems to solve, such as *security* and *dynamic binding*, since more and more the resources local to a system are likely to be open for access and execution to foreign entities.

These issues dramatically influence the application layer, since when executing an application its components have to accomplish their task by *cooperating* with other components in a distributed, heterogeneous, unpredictable, and possibly malicious execution environment. Moreover, the location of the application components is likely to change over time, so that their execution is no longer limited to one given site.

2.2 Interaction in Internet Applications

The very nature of the Internet, born as a heterogeneous collection of nodes, services, and information repositories, all asynchronously working as an ensemble, is intrinsically composite. Correspondingly, typical Internet applications are made up of a multiplicity of heterogeneous components, which have to work together in an distributed, heterogeneous, and unpredictable environment. This raises several issues which call for new abstractions and tools. For instance,

- in traditional distributed applications, each component is explicitly designed for interacting with other known components. Instead, unpredictability and open-ness make usually necessary that agents be able to dynamically acquire the capability of interacting with unfamiliar/previously unknown services and data;
- in traditional distributed applications, components have usually to exchange simple data structures for elaboration and synchronisation. Instead, the typical complexity of the tasks to be accomplished, and of the global application strategies, makes Internet agents be often in need of exchanging more complex information, such as composite documents, structured knowledge bases, and even planning strategies.
- traditional distributed applications typically restrict user interaction to few, simple input/output operations. Instead, the user-oriented nature of most Web technologies makes applications intrinsically open to a multiplicity of different interaction patterns.
- traditional distributed applications are typically based on specialised, possibly proprietary technologies, which are tailored to specific application requirements. Instead, Internet applications have to exploit standard technologies, like Web browsers and servers, and Java, and to be as much as possible open to new emerging standards.

Thus, managing interaction and access to resources and services is one of the key point in the design and development of Internet applications, and raises the issue of selecting the most suitable coordination model for a given application. In this context, mobility introduces some peculiar coordination issues.

2.3 Mobility of Internet Agents

Traditional distributed applications statically allocate active entities (processes) to given network nodes, and force them to cooperate by exchanging data structures usually marshalled to ASCII. Therefore, *mobility of data* is intrinsic to distributed computation and to the Internet itself, which provided facilities for data communication (eg. the ASCII code) since its birth. The advances in Internet technologies have broken the boundary of the fixed execution environment, thus leading to new kinds of mobility. *Mobility of code* permits data *and* programs to move and execute on demand on different sites (eg. applets). What we are interested here, however, is *mobility of agents*, where stateful entities can move through the Internet by carrying along their own internal state and behavior.

Mobility brings several immediate benefits:

- mobile agents can move to the data it needs to manipulate and save bandwidth;
- mobile agents execution can proceed without being influenced by the variable latency of the network or even in absence of network connection;
- mobile agents can "lately bind" part of their code, so as to implicitly adapt to their current local execution environment;
- intelligent mobile agents can deliberately adapt their behaviour to their current execution environment, and even moving to another one whenever the availability of the local resources does not match the component's needs and requirements;
- mobile agents provide the most suitable abstraction for mobile devices.

The above benefits, along with the fact that mobility of software components is intrinsically promoted by mobility at the physical layer, make it easy to predict that Internet applications will be more and more based on software components *(i)* exploiting mobility as a dimension of their computational life, and *(ii)* featuring different levels of "intelligence" [31]. Correspondingly, new coordination problems are raised: *(i)* how mobile and unpredictable software components can be organised to interact fruitfully and efficiently, and *(ii)* how to effectively support intelligence of the components, and possibly embed some level of intelligence in the coordination itself.

3 Coordination Models for Internet Agents

Coordination is managing the interaction among software agents [30]. A *coordination model* provides a formal framework in which the interaction of software agents can be expressed. Generally speaking, a coordination model deals with the creation and destruction of agents, their communication activities, their distribution and mobility in space, as well as the synchronisation and distribution of their actions over time.

More precisely, a coordination model consists of three elements [9]:

1. the *coordinables*, which are the entities whose mutual interaction is ruled by the model. These could be Unix-like processes, threads, concurrent objects, and even users.

2. the *coordination media*, which are the abstractions enabling agent interactions, as well as the core around which the components of a coordinated system are organised. Examples are the classic media like semaphores, monitors, or channels, or more complex media like tuple spaces, blackboards, pipes, etc.

3. the *coordination laws*, which define the behaviour of the coordination media in response to interaction events. According to [22], the laws can be defined in terms of a *communication language*, that is a syntax used to express and exchange data structures, and a *coordination language*, that is a set of interaction primitives and their semantics).

¿From a software engineering viewpoint, a coordination model should work as a source for design metaphors, abstractions, and mechanisms effectively supporting the development process of distributed, multiagent applications. The choice of a suitable coordination model is a crucial issue when dealing with software architectures allowing agent mobility over the Internet, where issues such as distribution, heterogeneity, unpredictability, integration with standard technologies, and so on, require some specific criteria for the choice of the most suited coordination model.

3.1 Internet Agent Coordination

In spite of the above considerations, among the several programming languages and systems aiming to support and facilitate the design and development of applications including mobile agents [27], only a few focus on the explicit definition of a coordination model.

Instead, in the context of both mobile agent systems and AI multi-agent systems, *direct* (peer-to-peer) *communication* is usually adopted as the main interaction protocol between application components. However, direct communication does not suit Internet agents, since it implies a strict coupling between interacting entities in terms of naming rules (who the partners are), space control (where the partners are) and time control (when to interact).

The interposition of a specialised communication middleware infrastructure, such as in CORBA, is not satisfactory either, since this approach does not provide any of the abstractions and metaphors needed to effectively support the design of complex applications based on Internet agents.

Meeting-oriented coordination models, implemented by the ARA [24] and MOLE [3] systems, constrain direct interactions to occur in the context of well-known meeting points. However, this only partially solves the above problems, given that agents still need to share the knowledge about the meeting point identity and location.

Instead, *tuple-based coordination models* seem particularly attractive, as discussed in the next subsection.

3.2 Tuple-based Coordination

This mainstream research field originates from the work on the Linda programming language [6]. Linda relies on the notion of *tuple space* as an *associative blackboard* used by active components to communicate, synchronise, and cooperate by reading, writing, and consuming *tuples*.

The tuple space model is especially attractive because it scales: multiple tuple spaces can be considered a coordination media populating a coordination space, the communication language is made up of tuples, and the coordination language provides coordinated entities with the primitives for tuple space access and modification. The coordination language includes the `out` operation, to write a tuple in a tuple space, and the `in` and `rd` operations, to send a tuple template and wait for the tuple space to return a tuple matching the template, respectively either deleting it or just copying it from the tuple space. Since the semantics of `in` and `rd` implies that a coordinable agent blocks until a matching tuple is found, the Linda coordination language provides for synchronisation, too.

The tuple space metaphor is best understood as a coordination model embedded in a conventional programming language. For instance, when Linda is embedded in Java, as in [11], coordinables are active Java objects, the coordination medium is a multiset of Java objects, the coordination laws are those describing the semantics of Linda-like primitives on reading, writing, and consuming Java objects in the tuple space.

Many extensions, derivations and implementations have followed the original model, trying to exploit and maximise the benefit of its many features, such as:

- cleanness and expressiveness of the communication language [7];
- separation between computation and coordination [6];
- generative communication [17], leading to spatial and temporal uncoupling between coordinables;
- associative access to communication information [10].

While Linda originates in the field of parallel programming, its underlying coordination model is well-suited also for open, distributed systems. For instance, the clean separation of the computation and coordination issues simplifies the design of complex software agents. Moreover, by uncoupling interacting entities, coordination models based on tuple spaces solve many problems related to mobility, since agents can interact without knowing who and where the partners are. Last but not least, associative access enables agents to retrieve data from the tuple space via some data-matching mechanism (like pattern-matching, or unification) integrated within the tuple space itself, and makes it easier to deal with data incompleteness as well as with dynamicity and heterogeneity which are typical of Internet nodes, when seen as information sources.

4 Coordination on the Web

The World Wide Web is currently the most popular platform to access Internet services, so it has the potential to become the standard infrastructure to build integrated applications. In fact, most application domains are turning to the Web as the environment of choice for building innovative applications leveraging on the open standards, their diffusion, and the programmable nature of the available services. For instance, there is a growing interest in *active document management systems* based on the Web infrastructure [16]. These applications typically exploit *multiagent* technologies, meaning that they are highly concurrent, distributed, and based on mobile code.

However, the Web in its current state does not provide enough support for those applications based on agent-oriented programming, like groupware or workflow, which require sophisticated agent coordination.

In fact, most Web-based applications are either server-centric (interfacing applications via CGI to a mainframe-like central server machine), client-centric (applets providing application services to users without a real distribution concept), or not integrated at all with the Web (applications whose user interface is implemented by applets or plug-ins connecting with some proprietary protocol to a proprietary server). All these approaches do not really satisfy the idea of a structured configuration of (autonomous) agents: they are either not really distributed, or not integrated with the Web.

Several issues arise when facing the problem of designing an effective Internet programming language. In fact, several design choices may be in principle adopted when defining the coordinables, coordination media, and coordination laws in a Web-based software architecture. More precisely:

- Which entities, in addition to application-specific agents, should become the coordinables of the architecture? In particular, what role should Web services assume w.r.t. the coordination architecture?
- Should the coordination architecture define a single or a variety of – possibly independent – tuple spaces? In the latter case, how should they be distributed and how can coordinables identify and refer to tuple spaces?
- Given an architecture including a coordination medium and some coordinables, what primitives should be included in the coordination language? Moreover, what communication language and primitives better suit the Web scenario?
- Should the coordination laws be fixed once and for all by the model or should they be adjustable at the application level? In the latter case, how can we specify the coordination laws?

In the rest of this section, we discuss the above design issues and survey some solutions recently proposed. In particular we will analyse the following proposals:

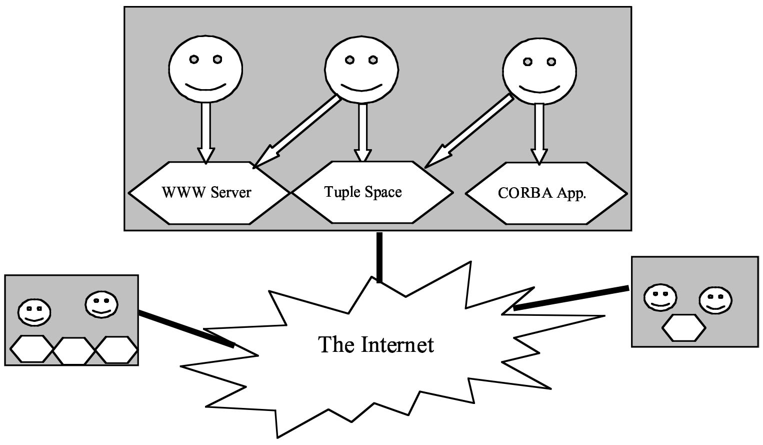

Figure 1: Tuple spaces as additional Internet services.

JavaSpaces [28], proposed and implemented by Sun Microsystems, T Spaces [32], developed by IBM, WCL [26], developed at the University of Cambridge, Lime [25], developed at the University of Washington, PageSpace [12], developed jointly at TU Berlin and at the University of Bologna TuCSoN [23], developed at the University of Bologna, and MARS [4], developed at the University of Modena. The main features of these systems are summarised in Table 1, and they will be analysed in more detail in the following. The analysis will be made by keeping in mind that and any proposal in this context should be well-integrated in the Web scenario and allow a simple integration with all the services and tools already available and widely used, such as WWW servers, browsers, CGI applications, and CORBA services.

4.1 Coordinables

The definition of a coordination model should start from the identification of the coordinable entities, i.e., the active entities accessing the coordination media.

Since we are interested in models including mobile entities, it is important to define first *non-mobile* coordinables. In fact, as stated in Section 2, even if we focus on applications based on mobile agents, in general an Internet application could include several non-mobile entities, such as WWW servers and CORBA-compliant services, which an agent may be in need to interact with in order to perform its task.

A possible choice is to consider all these non-mobile entities as simply server objects, by defining their coordination medium as merely another Internet service.

The Figure 1 depicts a situation in which agents can use indifferently as a coordination medium a Tuple Space, a WWW server, or a CORBA ORB.

However, this approach is not satisfactory. On the one hand, the language

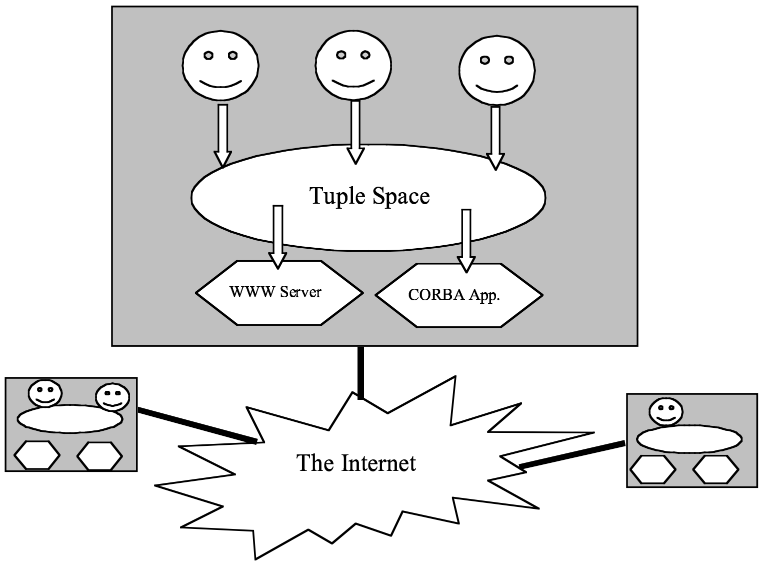

Figure 2: Internet services masked behind tuple spaces.

designer would be forced to provide a large number of mechanisms to enable agents to access not only tuple space services, but also any other service. On the other hand, such a proposal would not enforce an incremental approach to Internet-application construction, as it would complicate the definition of application agents. Admittedly, this solution is better for *Intranet* applications, which seldom require access to the "open" Web world, and mostly require coordination among agents specifically defined for a given application (for instance, this is the case with the WCL system).

As an opposite approach, the coordination architecture could be designed as to become "The Internet" service, masking and providing access to any existing service via the tuple space itself.

Figure 2 abstracts this situation, showing a tuple space used as main coordination medium among agents and Internet services.

This is the choice adopted by MARS, which associates a single global tuple space service to each node of the network able to accept and execute agents. MARS gives an agent the possibility of interacting with the execution environment, its resources and services only via the tuple space.

The above solution enforces a very simple programming style, and consequently leads to the definition of a very simple programming language, by making Linda-like coordination the only possible way of interaction in Internet applications. However, it may turn out to be not appropriate for all applications, as components (for example proxy-servers) may require to act as both clients and servers, thus making complex their integration into an application.

Between the above two extremes, a more suitable approach could conceive

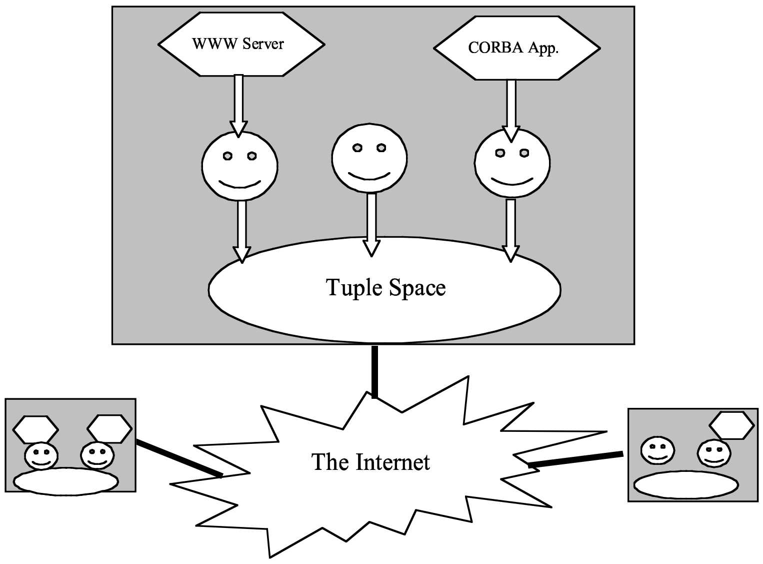

Figure 3: Tuple spaces as middleware.

the coordination model as a sort of middleware, selecting as coordinables all the entities populating the Web and offering a uniform underlying infrastructure. However, since legacy entities have not been explicitly designed to access a tuple space, their integration into the architecture may require special-purpose agents in charge of mediating and ruling the access to the tuple spaces from other entities.

Figure 3 depicts such situation, that is typical of the PageSpace system.

To the best of our knowledge, PageSpace is the only research proposal which explicitly addresses the problem of defining and characterising coordinables, i.e., Internet agents, and all the other classes of agents that must be part of model. Instead, both JavaSpaces and T Spaces only aim to define an architecture for the coordination medium, implicitly assuming the presence of coordinables, i.e., agents that can access the space. However, these proposals lack in identifying all the entities which are part of the coordination architecture and, then, do not properly face the problem of Web integration.

In most cases, the only explicit reference to this problem relates the presence of special-purpose agents, e.g., applets executing inside Web browsers, to enable access to a coordination medium via a browser [25, 23].

4.2 Coordination Media

A simple, not widely-distributed application may take advantage of a single tuple space, where all the application components can be coordinated in a centralised way. This may simplify the application design, and may not require

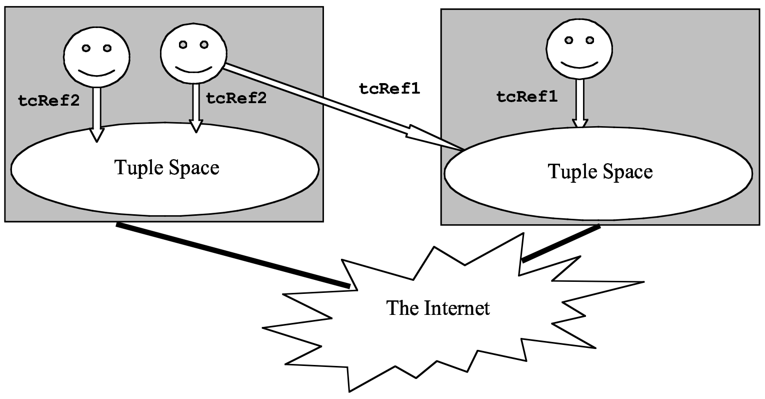

Figure 4: Transparent access to tuple spaces.

to take into account the issue of denoting tuple spaces, since there is just a single default tuple space. However, building widely-distributed and complex applications by using a single tuple space is not feasible.

On the one hand, this would create remarkable latency and reliability problems. On the other hand, this would not enforce any modularity principle, and would make difficult the application management. In this case, applications should be built around a multiplicity of tuple spaces, appropriately distributed in the Internet and accessed by agents according to their specific needs.

The availability of multiple independent tuple spaces enables de-centralisation and modularity, but introduces novel problems. In particular, a coordination model including multiple tuple spaces requires for agents a suitable way to denote and access the spaces themselves.

Many systems (e.g., JavaSpaces, PageSpace, and WCL) define an object-oriented tuple space model in which tuple spaces are objects themselves.

Figure 4 represents the fact that application agents can exchange tuple space references and use them to access a given tuple space, transparently to location.

This solution is very simple and intuitive, and provides for a transparent, location-independent way to access to tuple space. However, in the context of Internet applications, network-unawareness is not the most natural choice, as argued by [29]. In fact, enforcing transparency in a wide-area network does not lead to an efficient programming paradigm, because the unpredictable latency times would make the application performances hard to be predicted.

The WCL proposal addresses the problem of communication latency by providing for transparent migration of tuple spaces, in order to enforce locality of the accesses: either tuple spaces are migrated to the clients that need to access them, or clients can be migrated transparently to a given tuple space. This requires a complex run-time support, and it is not integrated in the Web at all.

In the presence of network-aware mobile agents, explicitly moving through a collection of different execution environments, a transparent choice is even more

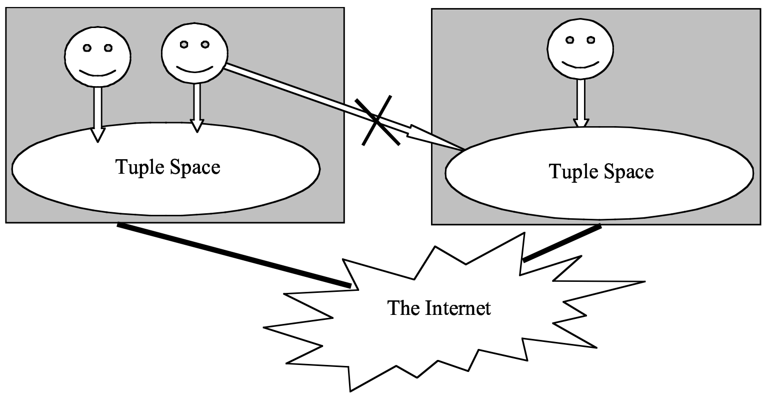

Figure 5: Implicit access to the local tuple space in MARS.

odd: if agents move to increase locality with the needed resources, enforcing a transparent, network-unaware access to the coordination media – without the possibility of knowing where the tuple space is – would void the agents' mission. The organisation of the tuple spaces, as well as the related way of accessing them, must take into account the network-aware mobility of agents. In this context, we can distinguish two main approaches: the implicit one and the explicit one.

The *implicit* approach alleviates the agent from the need of explicitly referring to a tuple space, and automatically lets the agent access a default tuple space, possibly depending on its current position in the Internet. Therefore, the implicit approach tightly connects the issue of agent mobility to the coordination model.

The MARS system adopts an implicit, context-dependent, approach to identify and access a tuple space

Figure 5 shows a single tuple space associated to each execution environment; a reference is bound to any incoming agent. The tuple space can be used to access to resources and services on that node, as well as to enable inter-agent coordination, in a implicit way, without needing to name it or to explicitly maintain a reference to it. When an agent migrates and arrives to a new site, a new connection is established with the tuple space of the new node, while the old connection gets lost. There is no way for an agent to access to a remote tuple space but by explicitly migrating to that node.

Lime adopts a different implicit approach: an agent carries a private tuple space during its movement, and this is the only tuple space it can access during its life (Figure 6).

As the agent arrives on a node, its private tuple space is automatically merged with the one associated to the execution environment. Then, the private tuple space of the agent can be used to implicitly access the resources of that node. In addition, different nodes can federate and merge their associated tuple

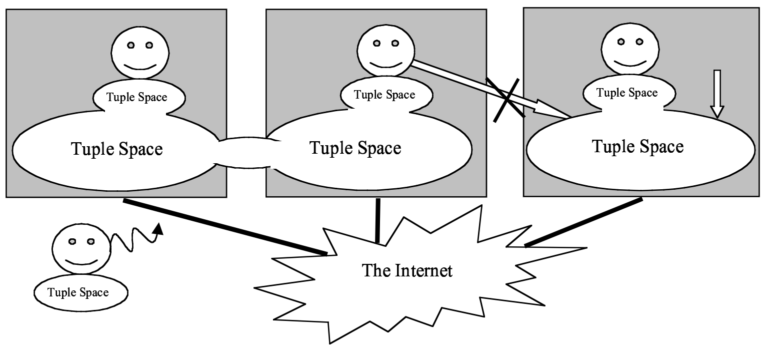

Figure 6: Implicit access to the agent-own tuple space in Lime.

spaces, thus allowing an agent to access to a remote resource via the federated tuple space.

The main drawback of the implicit approach is that it forces an agent to migrate to access a specific tuple space. Sometimes this is not the most efficient choice. For example, if an agent needs a single access to a given tuple space, it may be not appropriate to migrate the agent towards that node, and enabling a remote access would be preferable.

The *explicit* approach, while maintaining a network-aware architecture, enables remote access to tuple spaces by associating a global naming scheme to tuple space. The most natural choice for naming a tuple space would be a Web compliant one (i.e., URLs).

When adopting an explicit approach, the implicit approach has not to be necessarily discarded: although it is always possible to refer to a tuple space by its global identifier, the possibility for an agent to implicitly refer to a "default" tuple space, depending on its current location, may reduce the complexity of the agent and permit applications to be based on "context-dependent" coordination actions (Figure 7).

TuCSoN adopts the above approach and enables both implicit and explicit access to a tuple space.

4.3 The Coordination Language

When adopting a coordination model based on tuple spaces, defining the coordination language means to define the data types and language primitives to access the spaces.

With regard to the latter point, the original Linda model, mostly oriented to parallel programming, defines a simple language based on a limited set of typed primitives, to be used to compose tuples.

The current trend, followed by PageSpace, JavaSpaces, and MARS, is to define object-oriented tuple space models, in which both tuple spaces, as well as tuples and their component fields are Java objects. This means that a large

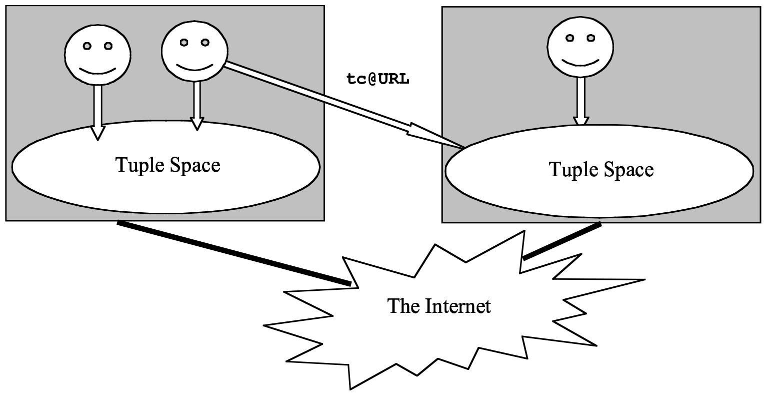

Figure 7: Implicit and explicit access to tuple space in TuCSoN'

number of different data items are in need to be managed and exchanged during coordination actions.

However, this may be not always the most appropriate language for all applications. Some specific application problems may require different typed coordination mechanisms. For instance, the T Spaces system is oriented towards the management of large data, and the adopted model is a relational one: a tuple type identifies a table schema, whereas a tuple is one row in a table. TuCSoN that is oriented to the coordination of intelligent information agents, defines a tuple space model in which tuples are terms, namely uninterpreded first-order logic predicates.

Concerning the communication primitives, most of the systems define a static and limited set of primitives to be used to access the coordination media. Typically, this set includes operations having the same semantic of Linda primitives, and a limited number of additional operations, such as collecting all the tuples matching a given template, as in MARS.

In the case of applications based on autonomous agents, asynchronous operations like notification mechanisms as in WCL and JavaSpaces are useful: an agent can require a tuple to a tuple space without being blocked waiting for a matching tuple. The agent goes on with its execution, and can retrieve a matching tuple later, when available. However, when mobile agents are involved, notify mechanisms (i.e., the tuple space signals an agent about the availability of a previously requested tuple) are not suitable, because they would require the run-time support tracking agents' movements to locate and notify them.

Due to the simplicity of Linda primitives, performing complex coordination on large set of tuples may be difficult and may require the composition of a large number of primitive operations. For this reason, T Spaces enables agents to to add any new primitive to a tuple space, in order to implement any needed operation on the tuples stored in it. However, this approach cannot generally suit Internet applications, based on autonomous agents, because either only the agent that has installed the new primitive can invoke it or some form of strict coupling between the agents has to be provided, to let them know which operation the other agents have installed in the tuple space.

4.4 Coordination Laws

Given a coordination medium, the coordination laws specify the behaviour of the medium itself in response to communication events, i.e., the invocation of communication primitives by one of the coordinables.

In the standard Linda model, the coordination medium embeds fixed, statically defined, and not modifiable coordination laws. In particular, these laws are based on the concepts of associative access, and make an input request for data be supplied with one tuple, if any, matching on the basis of a pattern-matching mechanism with a specified template.

Since its first definition, different extensions to the Linda model have been proposed defining different coordination laws, often directly driven by the adopted data model for tuples. In the context of Internet coordination systems, the object-oriented tuple space model adopted for instance by JavaSpaces substitutes the classical Linda pattern-matching mechanism by making two objects match each other if their serialised forms are equals. Instead, logic unification is assumed as the basic matching mechanism in the logic-based coordination space in the TuCSoN model [14].

Independently from the adopted coordination laws, most of the tuple space systems presented in the literature adopt the Linda approach of embedding fixed coordination laws into the coordination medium. The same choice has also been adopted by most of the proposals we discussed in the context of Internet-oriented coordination, including PageSpace, WCL, Lime, and JavaSpaces. However, in a wide and unpredictable word as the Internet, this choice may not be the most appropriate one if mobile agents roaming through heterogeneous execution environments have to interact.

Other proposals, such as T Spaces, TuCSoN and MARS, conceive the coordination medium as a configurable kernel. The idea is to make the coordination medium configurable, by allowing new coordination laws to be defined and embedded in it. This choice is driven by the consideration that Linda-like coordination lacks flexibility and control, since interactions are constrained by the built-in data-access mechanisms provided by the tuple space. Thus, any coordination policy not directly supported by the model has typically to be programmed inside the agents. In an open, heterogeneous and unreliable environment as the Internet, this is likely to notably increase the agent complexity, which are forced both to implement in their code the peculiar coordination protocols required by an application and to solve Internet related issues, such as heterogeneity and dynamicity of the information sources on the execution environments. In addition, this makes somehow coordination rules distributed between blackboards and the agents, thus affecting the global application design and breaking the logical separation between coordination and algorithmic issues.

¿From this perspective, T Spaces make it possible to change the basic associative mechanisms, and to define new communication primitives, which, as already stated, seems not well suit an open Internet scenario. Instead, TuCSoN and MARS forbid both the introduction of new primitives and the modification of the matching mechanism, but provide full programmability of the coordination laws by enabling the definition of a new behaviour in response to communication events. In particular, in TuCSoN the coordination media are suitable to be programmed by intelligent, meta-level agents, which could then be exploited to dynamically tune the global behaviour of a coordinated application.

In its turn, MARS, coherently with its OO tuple space model, follows an object-oriented approach to coordination laws programming: a specific method of a specific interface class can be implemented and associated to specific communication event. The method overrides the default OO associative mechanism and implements the specific behaviour the tuple space should exhibit at the occurrence of the associated communication event. The above characteristics make the MARS tuple space programmability suitable to high-level service and system management.

5 Discussion

In the above section, we identified several design solutions and the pros and cons of the different solutions in different contexts. Starting from those considerations, we can summarise which design solutions better suit a scenario where tuple-space-based coordination models rule the Web.

Coordinables
A coordination model for the Web must precisely define the coordinables of the model. In particular, an appropriate role must be assigned to Internet services to enable a real integration of the model with the Web. The PageSpace proposal for a tuple-based middleware, where any Internet service can find its specific role as special-purpose coordinable (possibly with the mediation of specifically designed agents) is the only concrete attempt in this direction. PageSpace, without having the ambition to provide a definitive answer, identifies a suitable direction towards an engineered approach to Internet applications development. However, the option of masking Internet services behind a tuple space (instead of defining them as coordinables) should not be *a priori* excluded, since it may be useful to provide a simple tuple-based access to already existing services.

Coordination Media
A coordination architecture on the Web should be based on a multiplicity of independent tuple-space abstractions, distributed across Internet sites and possibly independently managed. Application agents can take advantage of a network-aware architecture, in which they can implicitly refer to tuple spaces depending on their current position. In addition, they could also be given the possibility of explicitly referring to a given tuple space, independently from their current position, by explicitly locating it via an URL, as in TuCSoN.

A transparent, network-unaware choice, making agents use generic references, should be generally discouraged in the case of Internet applications. However, small-size network domains, i.e., intra-organisation ones, can take advantage of transparency when opening to the execution of Internet agents. In fact, a transparent coordination architecture can be useful to distribute internal resources (as in WCL) and possibly by controlling intra-domain agent migrations, in order to achieve the best internal resources structure without forcing application agents to be aware of the internal organisation of the domain. An interesting alternative solution for intra-domain coordination management which is worth to be evaluated can be inspired by Lime: if the tuple spaces of different sites can merge together, this can enable an agent to access transparently the resources of other nodes, as if they were local.

Coordination Language
In general a coordination model should always be kept as simple as possible, in order to achieve a simple and clean integration within a programming language. A small, Linda-like set of coordination primitives is usually sufficient for most application needs, and we discourage the adoption of complex coordination languages with a lot of primitives, each devoted to solve a peculiar problem. Similarly, we feel that the coordination language should not be dynamically extensible (by integrating new primitives not originally defined by the coordination model, as in T Spaces), since this makes application management more complex and limits agent uncoupling.

Instead, if new functionalities have to be added to the coordination model, a better solution consists of embedding them as coordination laws in the media, without affecting the coordination language.

A design choice that is more difficult to evaluate relates to the data model for tuple spaces. Among several models (such as object-oriented tuple spaces, logic tuple-space, relational tuple-space), a single model cannot answer to all application needs. The best solution would be to define a coordination architecture in which different tuple spaces based on different data models could co-exist under the same coordination language, so that the adoption of multiple tuple spaces would not only be a way to better modularise and distribute coordination activities, but also a way to define special-purpose coordination media to be used depending on application needs.

Coordination Laws
While the simplicity of the Linda coordination language well suits Internet applications, constraining coordination laws to the built-in data-access mechanisms provided by tuple spaces can be sometime to restrictive: any coordination policy not directly supported by the model has to be charged upon agents. Therefore, a programmable tuple space model should be integrated into a Web coordination architecture: with no change to the coordination language, one should be able to tune the behaviour of the tuple space in response to agents' invocation of communication primitives to the specific needs of both the application agents and the execution environment. Also in this context, programmable coordina-

tion media based on different programming models and languages, let them be the logic programming model provided by TuCSoN or the object-oriented one provided by MARS, should be allowed to co-exists, to meet the needs of different application areas.

6 Conclusions and Open Directions

The power of tuplespace-based coordination models makes them a suitable technology to enable and facilitate the design and development of Internet applications where mobile agents spread over a Web scenario. Such a power is confirmed by the number of proposals that, to different extents and following different approaches, aim to define and integrate the Web with Linda-like coordination models and languages.

In this paper we tried to identify the language design solutions that can better suit the current Internet scenario. However, several other research issues, not relevant for the systems surveyed and (consequently) neglected by this paper, have still to find an appropriate solution before these technologies become practical for Internet languages, and an effective component of Internet applications.

As a first consideration, it is clear that any proposal should take into consideration any existing standard as well as emerging ones in order to gain acceptance. Therefore, since the forthcoming XML technology is likely to become a de-facto standard for data structure representation and exchange across the Internet world, any coordination model should become XML compliant, not only for data exchange, but also for any coordination related issue. In fact, it is possible to represent in XML not only data but also agents and services, as well as high-level interaction protocols. This by no means implies that XML is likely to mine the role of a coordination model. Instead, a coordination model integrating the XML technology, both at the architecture and at the language level, could provide a framework where a wide range of agent-based proposals in the area of Internet computing will find a coherent role.

Another promising research direction concerns security, which is an issue intrinsically connected to coordination. In fact, while coordination technologies for the Internet aim at enabling interaction and making it fruitful, security is meant to control and limit interaction, so to make it harmless [13]. In other words, while a coordination language may allow an agent to perform any communication primitive and access any available data structure, a security policy would instead limit the interaction protocol, for instance by forbidding the access to some resource or communication media. A coordination framework for an Internet programming language should enable the application-level management of security policies, to achieve the best compromise between expressive power and safety in interactions. Still, the research communities of coordination and security are separated worlds and a few projects have already explicitly addressed in a uniform way coordination and security issues.

The Internet is likely to be soon populated by agents, roaming through the nodes, accessing structured documents and services, and therefore consuming resources. Consequently, market-oriented techniques for assigning prices to resources, currency exchange mechanisms as well as techniques to rule the accesses and controlling the consumption of Internet resources will become necessary. Otherwise, the Internet would soon experience the problem known as "*tragedy of the commons*", i.e., a global degradation in the quality of Internet services, whatever the amount of available resources [18]. In this respect, a coordination model should also take into account economic issues when enabling and ruling interactions, such as assigning prices to interaction events, dealing with limited resources (possibly with the help of auction-based protocols), handling virtual currencies carried by agents. These issues cannot be solved simply adopting a suitable run-time support, but will influence the application level, strongly impacting on the coordination model and language.

Last but not least, we remark that no Internet programming language, in itself, in sufficient for developing large and complex Internet applications. Instead, as already recognised by the software engineering community, high-level tools and suitable design patterns are needed. Some preliminary works on design patterns including mobile agents are [2, 20]. However, to be effective, any possible definition of a set of patterns for Internet agents requires some precise assumptions on the underlying software support and, in particular, on the coordination model and its architecture.

Acknowledgments: This paper has been partially supported by Italian MURST and by a grant of the Univ. of Bologna. We would like to thank Davide Rossi for his helpful comments.

References

[1] Inc. Ad Astra Engineering. Jumping beans v.1.04, 1998. `http://www.jumpingbeans.com/`.

[2] Y. Aridor and D. Lange. Agent design pattern: Elements of agent application design. In *Proc. Int. Conf. on Autonomous Agents*. ACM Press, 1998.

[3] J. Baumann et al. Communication Concepts for Mobile Agents Systems. In K. Rothermel and R. Popescu-Zeletin, editors, *Proc. First Int. Workshop on Mobile Agents*, volume 1219 of *Lecture Notes in Computer Science*, pages 123–135, Berlin, 1997. Springer-Verlag, Berlin.

[4] G. Cabri, L. Leonardi, and F. Zambonelli. Reactive Tuple Spaces for Mobile Agent Coordination. In K. Rothermel and F. Hohl, editors, *Proc. 2nd Int. Workshop on Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, pages 237–248, Stuttgart, Germany, 1998. Springer-Verlag, Berlin.

[5] L. Cardelli and A. Gordon. Mobile Ambients. In M. Nivat, editor, *Proc. of Foundations of Software Science and Computation Structures (FoS-*

SaCS), European Joint Conferences on Theory and Practice of Software (ETAPS'98), volume 1378 of *Lecture Notes in Computer Science*, pages 140–155, Lisbon, Portugal, 1998. Springer-Verlag, Berlin.

[6] N. Carriero and D. Gelernter. Coordination Languages and Their Significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[7] N. Carriero, D. Gelernter, T. Mattson, and A. Sherman. The Linda Alternative to Message-passing Systems. *Parallel Computing*, 20:633–655, 1994.

[8] D. Chess et al. Itinerant Agents for Mobile Computing. *IEEE Personal Communications*, 2(5):34–49, October 1995.

[9] P. Ciancarini. Coordination Models and Languages as Software Integrators. *ACM Computing Surveys*, 28(2):300–302, 1996.

[10] P. Ciancarini and D. Gelernter. A Distributed Programming Environment based on Multiple Tuple Spaces. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, pages 926–933. Institute for New Generation Computer Technology, Tokyo, 1992.

[11] P. Ciancarini and D. Rossi. Jada: Coordination and Communication for Java agents. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems: Towards the Programmable Internet*, volume 1222 of *Lecture Notes in Computer Science*, pages 213–228. Springer-Verlag, Berlin, 1997.

[12] P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, and A. Knoche. Coordinating Multiagent Applications on the WWW: a Reference Architecture. *IEEE Transactions on Software Engineering*, 24(5):362–375, 1998.

[13] M. Cremonini, A. Omicini, and F. Zambonelli. Modelling network topology and mobile agent interaction: an integrated framework. In *Proc. 1999 ACM Symposium on Applied Computing (SAC'99)*, S. Antonio, Tx, 1999.

[14] E. Denti, A. Natali, and A. Omicini. On the Expressive Power of a Language for Programming Coordination Media. In J. Carroll et al., editors, *Proc. ACM/SIGAPP Symp. on Applied Computing (SAC 98)*, pages 169–177. ACM Press, 1998.

[15] G. DiMarzoSerugendo, M. Muhugusa, and C. Tschudin. A Survey of Theories for Mobile Agents. *World Wide Web*, 1(3):139–153, 1998.

[16] A. Fuggetta, G. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.

[17] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[18] A. Gupta, B. Jukic, M. Parameswaran, D. Stahl, and A. Whinston. Streamlining the Digital Economy: How to Avert a Tragedy of the Commons. *IEEE Internet Computing*, 1(6), Nov.-Dec. 1997.

[19] N. Karnik and A. Tripathi. Design Issues in Mobile-Agent Programming Systems. *IEEE Concurrency*, 6(3):52–61, 1998.

[20] E. Kendall, P. MuraliKrishna, C. Pathac, and C. Sureash. Patterns of intelligent and mobile agents. In *Proc. Int. Conf. on Autonomous Agents*. ACM Press, May 1998.

[21] J. Kiniry and D. Zimmerman. A hands-on look at Java mobile agents. *IEEE Internet Computing*, 1(4):21–33, July– August 1997.

[22] A. Omicini. On the semantics of tuple-based coordination models. In *Proc. 1999 ACM Symposium on Applied Computing (SAC '99)*, S. Antonio, Tx, 1999.

[23] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, (to appear), 1999. Special Issue on Coordination Mechanisms and Patterns for Web Agents.

[24] H. Peine and T. Stolpmann. The architecture of the Ara platform for mobile agents. In *Mobile Agents '97*, volume 1219 of *Lecture Notes in Computer Science*, pages 50–61. Springer-Verlag, Berlin, 1997.

[25] C. P. Picco and G. Catalin-Roman. Lime: Linda meets mobility. In *Proc. Internation Conference on Software Engineering*, page (to appear). IEEE Computer Society Press, 1999.

[26] A. Rowstron. WCL: A co-ordination language for geographically distributed agents. *World Wide Web*, 1(3):167–179, 1998.

[27] T. Thorn. Programming Languages for Mobile Code. *ACM Computing Surveys*, 29(3):213–239, 1997.

[28] J. Waldo et al. Javaspace specification - 1.0. Technical report, Sun Microsystems, March 1998.

[29] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. In *Mobile Object Systems*, volume 1222 of *Lecture Notes in Computer Science*, pages 49–64. Springer-Verlag, Berlin, 1997.

[30] P. Wegner. Interactive Software Technology. Technical Report CS-96-20, Dept. of Computer Science, Brown Univ., 1996.

[31] M. Woolridge and N. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[32] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T spaces. *IBM Systems Journal*, 37(3):454–474, 1998.

Table 1: Coordination Systems for Internet Agents

Systems	*Coordinables*	*Coordination Media*	*Coordination Language*	*Coordination Laws*
JavaSpaces	*Java Agents* - No explicit assumptions on the role of Internet services	Transparent architecture, explicit (network-unaware) access to tuple spaces (OO referencing)	OO (Java) Linda	Fixed
Lime	*Mobile Agents* - No explicit assumptions of the role of Internet services	Network-Aware architecture, implicit access to tuple spaces: agents access a private tuple space, which can be transparently merged with other tuple spaces on the current node of execution	Linda	Fixed
MARS	*Java Agents* - All services are masked behind the tuple spaces and accessed via them	Network-Aware architecture, implicit access to tuple spaces: an agent can only access the tuple space associated to its current node of execution	OO (Java) Linda	Programmable (Java reactions to communication events)
PageSpace	*Agents+Services* - Tuple spaces act as a middleware and Internet services becomes coordinables too, with the intermediation of special purpose agents.	Transparent architecture, explicit (network-unaware) access to tuple spaces (OO referencing)	OO (Java) Linda	Fixed
T Spaces	*Agents* - No explicit assumptions on the roles of Internet services	Transparent architecture, explicit (network-unaware) access to tuple spaces via (OO referencing)	Linda + User-defined Primitives	Programmable (primitives overriding)
TuCSoN	*Information Agents* - Some Internet services can be masked behind the tuple spaces and accessed through them	Network-Aware architecture, both implicit and explicit access: an agent can implicitly access the tuple space associated to its current node of execution and can also explicitly access, via URL, a remote tuple space	Logic-oriented (Prolog terms) Linda	Programmable (Prolog reactions to communication events)
WCL	*Agents, Application Space*	Transparent architecture, automatic re-allocation of agents and tuple spaces	Linda + Asynchronous Primitives	Fixed