Engineering Mobile Agent Applications via Context-dependent Coordination

Giacomo Cabri, Letizia Leonardi
Dipartimento di Scienze dell'Ingegneria – Università di Modena e Reggio Emilia
Via Vignolese 905 – 41100 Modena – ITALY
Phone: +39-059-2056133 – Fax: +39-059-2056126
{giacomo.cabri, letizia.leonardi}@unimo.it

Franco Zambonelli
Dipartimento di Scienze e Metodi dell'Ingegneria – Università di Modena e Reggio Emilia
Via Allegri 13 – 42100 Reggio Emilia – ITALY
Phone: +39-0522-430632 – Fax: +39-0522-496466
franco.zambonelli@unimo.it

Abstract: The design and development of Internet applications, requiring dynamic and possibly mobile access to Internet resources, can take advantage of an approach based on autonomous mobile agents. However, mobility introduces peculiar issues related to the modeling and management of the agents' coordination activities. This paper introduces context-dependent coordination as a framework for the design and development of Internet applications based on mobile agents, and shows how it can be supported by a proper coordination infrastructure. Context-dependent coordination is centered on the notion of programmable coordination media, as the software abstraction via which an agent, in an Internet site, can access to local resources and coordinate with local agents. Programmability stems from the fact that the behavior of the media can be fully configured to influence agents' coordination activities. This enables local administrators to configure coordination media so as to enact site-dependent coordination policies, and mobile agents to configure the accessed coordination media to obtain an application-dependent behavior of the media themselves. Several application examples shows that exploiting context-dependent coordination promotes a clear separation of concern in design and development, and can make applications more modular and easier to be maintained. The MARS system is assumed as an exemplar coordination infrastructure to clarify the concepts expressed and to show their actual implementation.

Index Terms: Mobile Agents, Internet Computing, Coordination Architectures, Internet Infrastructures, Agent-oriented Software Engineering, Context-aware Computing.

1 INTRODUCTION

The design and development of Internet applications introduce new problems and requirements over traditional approaches to distributed program development. On the one hand, decentralization, wideness, and openness of the Internet scenario, and the consequent partial knowledge available about it, require application components in charge of accessing its resources 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, suitable abstractions and tools are needed to model and enable effective and reliable worldwide access to Internet data and services by mobile users and mobile devices, other than by application components.

Autonomous mobile agents are a suitable approach to deal with the above issues in a natural and uniform way. In fact:

- by conceiving and developing applications in terms of autonomous entities, i.e., agents, executing in a proactive way to achieve a specific task and capable of reacting to changes in the environment, one can deal in a more natural and modular way with the dynamics and unpredictability of the Internet scenario [ZamJW01, Jen00];
- mobility is a useful technology to make agents dynamically transfer their execution to the sites where the resources they need are located, and access them in a local and reliable way [Whi97, FugPV98, KarT98]. Moreover, agent mobility can be useful to support nomadic users [BelCS01], and to model the presence in applications of mobile and wireless devices [Sat01].

However, mobility of agents introduces peculiar issues related to the modeling and the handling of the agents' coordination activities, i.e., interacting, communicating, and synchronizing both with other agents and with Internet resources. In particular:

- mobility at the Internet scale discourages the adoption of those coordination models relying on global communication abstractions and enforcing location-transparency [Wal97, CabLZ00a];
- administrators must have the capability of controlling and constraining the coordination activities of the agents that execute in their sites;
- agents should proceed with their execution and with their coordination activities accordingly to their own application-specific needs, despite the different characteristics and policies of the accessed Internet sites.

Starting from the above considerations, this paper introduces context-dependent coordination as a

framework for the design and development of Internet applications based on mobile agents. The main aim of the framework is to provide designers and developers with a uniform set of abstractions and infrastructures suitable for the engineering of mobile agent applications and of their coordination activities.

Context-dependent coordination is centered on the key notion of *local* and *programmable coordination media*. Coordination media, each associated to an Internet site, represent the means (i.e., the communication abstractions) via which agents, on a site, can access local resources and can interact with other local agents. Locality of coordination media promotes locality and scalability in interactions, and also provides agents with a perception of the Internet as a multiplicity of independent sites (*execution environments* [FugPV98]), a perception that suits the mobile nature of agents [Wal97]. In addition, programmability of the coordination media enables to configure them so as to influence agents' coordination activities, i.e., to enact specific coordination policies (*coordination laws*) to rule them. Such feature can be exploited to make coordination activities be adapted to the context in which they occur:

(i) *environment-dependent coordination*: administrators can configure the behavior of local coordination media to enact environment-specific coordination laws supporting and/or constraining the activities of the agents executing in the context of their administrative domains;

(ii) *application-dependent coordination*: application agents can program on-the-fly the accessed coordination media to enact application-specific coordination laws in the context of their specific application, whatever the environment in which they are currently executing.

Relying on programmable coordination media both as a design abstraction and as an actual middleware infrastructure promotes a clear separation of concerns in design and development, respectively. In fact, it enables to independently design and develop *(i)* agents, *(ii)* the application-specific coordination laws, and *(iii)* the environment-specific coordination laws. This paper shows, also with the help of several application examples, that context-dependent coordination can simplify and make more modular the design of mobile agent applications, and can make them easier to be developed and maintained. The MARS system [CabLZ00b] is assumed as an exemplar middleware infrastructure to clarify the concept expressed and to show their actual implementation.

The following of this paper is organized as follows. Section 2 focuses on agent mobility and on the need of suitable models and infrastructures to enable coordination in the presence of mobility. Section 3 details the key issues and concepts underlying context-dependent coordination and

programmable coordination media, and describes the MARS coordination infrastructure. Section 4 shows how context-dependent coordination can be effectively exploited in different application areas, to ease the management of several coordination problems typical of agent-based applications. Section 5 shows the positive impact of context-dependent coordination in the design and development of Internet applications. Section 6 discusses related work in the area. Section 7 concludes the paper and outlines research directions.

2 COORDINATION IN THE PRESENCE OF MOBILITY

Agent mobility can appear in different forms in today's Internet computing scenarios, and suitable abstractions and infrastructures are required to uniformly model and support the coordination activities of all types of mobile agents.

2.1 Types of Agent Mobility

Actual agent mobility. In most of the literature, agent mobility mainly refers to the capability of an autonomous software agent of dynamically transferring its execution, i.e., its code, data, and execution state, towards the nodes where the resources it needs to access are located. Despite the technological challenges that actual agent mobility introduces (in terms of security and resource control), its wise exploitation can provide for saving network bandwidth and for increasing both the reliability and the efficiency of the execution [Whi97, KarT98]. In addition, agent mobility can be effectively exploited to have nomadic users be assisted by personal software agents capable of following them in their activities [BelCS01].

Virtual agent mobility. In agent-based Internet applications, even those agents that do not actually transfer their execution can be perceived as mobile, for the sake's of application modeling. In fact, it is widely recognized that for Internet applications it is not suitable to enforce the principle of distribution transparency typical of distributed operating systems. Instead, it is more suitable to abstract the Internet as a multiplicity of networked *execution environments* (e.g., Internet nodes or administrative domains of nodes) [Wal97, CarG00, FugPV98] and to design applications in terms of *network-aware* agents, i.e., agents that are aware of the distributed nature of the target and that explicitly locate and access Internet resources in the environment in which they are located (e.g., via RMI binding to a URL-addressable resource). This implies a sort of virtual mobility of agents across execution environments.

Physical agent mobility. Mobile and wireless computing devices (better, the software processes running on them) connecting to the Internet from dynamically changing access points and interacting with its resources can be perceived, from the viewpoint of the fixed network infrastructure, in terms of autonomous and mobile software agents, and can be modeled as that [PicMR00, Sat01].

2.2 From Global to Local Coordination

The design and development of Internet applications based on mobile agents – whatever the specific type of mobility – discourage the generalized adoption of those coordination models such as peer-to-peer message-passing, client-server, distributed shared memories, relying on the presence of a globally shared communication abstraction. In fact:

- global interactions clash with the principle of locality that have always to be enforced in the design and development of complex software systems and that, in the case of mobile agents, calls for geographical locality in interactions and for communication abstractions supporting the agents' perspective of the Internet as a multiplicity of execution environments [Whi97];
- the globality of the Internet scenario and, therefore, the wide space of possible interactions, can make communications between mobile agents inefficient and unreliable, also calling for complex infrastructure to deal with message forwarding, and lookup of agents' positions and names [MurP99].
- the intrinsic autonomy and dynamics of mobile agents clash with the strict coupling (in terms of synchronization of the activities and/or necessity of sharing a common name space) usually enforced by global communication abstractions [CabLZ00a];

The suitable solution is to make agent coordination rely, whenever possible, on an infrastructure enforcing local and uncoupled interactions via a multiplicity of independent *local coordination media*.

A coordination medium is a coordination service typically associated to an Internet node or to a local administrative domain of nodes, in charge of acting as a mediator for all coordination activities in that site (node or domain). The coordination medium provides to local executing agents, via a specific API, the capability both to access the local resources of a site (*agent-environment coordination*) and to interact with other local agents and other application agents (*inter-agent coordination*), possibly in an asynchronous and anonymous way. A coordination medium can be based on any of a multiplicity of possible coordination model, e.g., meetings [Whi97], event-

channels [Bau98], or tuple spaces [AhuCG86], and has to provide the corresponding API to agents. For instance, in the case of coordination media based on the tuple space model, widely recognized as a suitable one for Internet applications [Cia98, FreHA99], agents are provided with primitives for reading, extracting, and storing tuples in a local tuple space, and all coordination activities take place locally via exchanges of tuples and synchronization over tuple occurrences.

Relying on local coordination media, both for application design and development, suits the presence of mobility. On the one hand, the locality and independency of coordination media intrinsically promote locality in interactions and scalability. On the other hand, the abstraction of coordination media can promote a uniform treatment of all types of mobility, by giving designers the uniform perspective of mobile agents as entities that move and interact across coordination media, disregarding whether such movement is virtual, actual, or physical. In addition, by promoting mediated and uncoupled interactions, coordination media can give an agent the possibility of indirectly interacting with other agents disregarding their current positions and names. This suits both agents' autonomy and openness of the scenario.

Of course, we are aware that our proposal, as well as any proposal in the Internet age, should provide for ease of integration in the established scenario to be feasible. At a first thought, this can be hard to imagine, since current Internet services and applications have no support for tuple spaces or, more generally, for local coordination media. However, we emphasize that our proposal, as it is the case of any new proposal in the Internet area, does not requires all of the existing Web sites and Internet services to suddenly support tuple spaces or whatsoever coordination media to be practical. Rather, one can envision specific groups (e.g., large ITC network enterprises, research networks, or spontaneous net-communities) to start adopting local coordination media for specific types of agent-based applications, possibly driving a wider acceptance of the model. This is what happened with the Napster and Gnutella communities, which rose up the attention on peer-to-peer models, and what the UE Agentcities Network aims at making happen with regard to agent-based Internet computing [Acities]. In the case of our specific proposals, the acceptance of the model can be further facilitated by the fact that it can be easily integrated in the current Web scenario. In fact, it is always possible to provide coordination media with standard API interfaces, e.g., as SOAP, WSDL, or Jini. Thus, coordination media can be accessed, other than by mobile agents, by any other Internet application in need to exploit a coordination service promoting locality in interactions (and context-dependent coordination, as detailed in the following).

2.3 A Case Study

As an application example (which will be assumed as a case study and will be further detailed through the paper), let us consider an application in which an actually mobile searcher agent roams an itinerary of Internet sites to access and retrieve information about specific resources, e.g., HTML pages or mp3 files. The searcher agent arrives at a site, locally retrieves the needed information, and continues its itinerary.

Agent-environment coordination. It is possible to think at a local coordination media as the means sites make available to agents to access local information. For instance, by assuming the presence of tuple spaces as local coordination media, site administrators can put in their local tuple spaces tuples such as (filename,extension,keyword,modification_time,size) to describe the local public files. A searcher agent on a site, by its side, can retrieve information about, e.g., HTML files related to the argument "coordination", by asking for tuples matching: (string?,"html","coordination",date?,int?).

Inter-agent coordination. Let us suppose that the searcher agent, during its roaming activity, discovers the presence of other interesting sites (e.g., via HTML links) not originally in its itinerary. At this point, the agent can create clones of itself (i.e., by spawning new agents with the same code and goals) and delegate the clones to go to those sites. This generates a tree of searcher agents, and makes it likely for two agents to arrive at 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 interactions between them is not feasible without the availability of complex communication middleware or without the adoption of odd design choices. Instead, a local infrastructure enabling uncoupled interactions, e.g., one based on tuple spaces, can be exploited in a very simple and elegant way. When a searcher agent arrives at 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.

The case study, despite conceived for actually mobile agents, can seamlessly extended to consider virtually mobile agents (in the case searcher agents access Internet sites from a fixed computer) as well as physically mobile agents. In the latter case, one can think at a searcher agent executing on a PDA, via which a roaming user can access information geographically distributed in a physical space.

3 CONTEXT-DEPENDENT COORDINATION

Enabling coordination via local coordination media is not enough. In fact, the coordination problems to be faced by agents requires the capability of ruling the coordination activities of mobile agents depending on the context in which such activities occur.

3.1 Issues of Context-Dependency

A coordination infrastructure promoting local and uncoupled interactions via a multiplicity of independent local coordination media fits agent mobility (both virtual, actual, and physical), and also invites thinking applications in terms of mobility. When an agent migrates to a new execution environment, its coordination medium changes accordingly. Thus, even if local coordination media are all based on the same coordination model, the agents' perceivable world changes as a consequence of their movement: what an agent 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-dependency is intrinsic in agent mobility and in the adoption of independent local coordination media. However, it is likely to be very complex to be handled by agents that are roaming the Internet to achieve a specific task. First, accessing data and services and coordinating with other agents in a foreign execution environment may require facing all the typical problems of open software systems, such as heterogeneity of languages and protocols, and opportunistic behavior in interactions [Jen00, ZamJW01]. For instance, with regard to the case study, it may not be always immediate for a searcher agent to discover the format in which information is stored on a site. Second, each execution environment has its own specific characteristics and may be in need of constraining – for security or resource-control reasons – the behavior of an agent in accessing the local resources and in coordinating with other local agents [FerG98]. With regard to the case study, agents may have to deal with specific constraints in accessing resources in the sites they visits, such as a limit in the amount of retrievable documents. Finally, agents that are part of a specific multi-agent application may require their coordination activities to occur according to specific application needs, despite the different characteristics of each local environment [OmiZ99, CabLZ00a]. For instance, in the case study, searcher agents may be in need to cope with more complex coordination protocols and to exchange more information other than the one aimed at avoiding multiple visits on a site.

To solve the above problems, a variety of heterogeneous and ad-hoc solutions have been

proposed. From the execution environment point of view, such solutions typically involve the introduction of specialized middleware components to facilitate (as well as to constrain) the access to specific local resources [BelM01], and of special-purpose agents to control inter-agent interactions [Pay02]. From the application point of view, such solutions typically lead to huge agents, requiring costly maintenance activities as soon as their coordination requirements and/or the external conditions (i.e., the execution environments) change [Klu01].

The alternative we propose is to rely on a coordination infrastructure in which the coordination media themselves are in charge of handling all the problems related to the local coordination activities, both agent-environment or inter-agent ones. Such an approach can provide a clean and conceptually simple framework to face coordination issues in a uniform way, whether these issues relate to environment-specific or application-specific coordination needs. However, this requires coordination media to be somehow active and to enable a dynamic adaptation of their behaviors in response to interaction events.

3.2 Programmable Coordination media

Whatever the model it relies upon, a coordination medium is generally characterized by: *(i)* a set of primitive operations to let agents access it and *(ii)* an internal behavior, intended as the computational activity performed inside the coordination medium in response to interactions events, i.e., in response to the invocation of a specific primitive performed by an agent.

Most of the existing coordination infrastructures fix the internal behavior of coordination media once and for all: the behavior of a coordination medium in response to a given interaction event is always the same. Some coordination infrastructures enrich coordination media with the capability of associating complementary actions in response to specific events, such as notifying external agents about event occurrences [FreHA99], but do not enable overriding the default behavior of the media.

An infrastructure based on *programmable coordination media* [DenNO98, CabLZ00b, MinU00] makes it possible – without changing the set of primitive operations used by agents to access the coordination media – to program the internal behavior of a coordination medium and override its default behavior so as to adapt it to the specific needs of applications or of the local environment. To this end, one must: *(i)* fully characterize the kind of access event of interest, in terms of the identity of the agent performing it, the primitive used to access the coordination medium, and the parameters possibly supplied in the invocation of the primitive; *(ii)* express the new behavior (we usually called

it “reaction”) to be assumed by the coordination medium in response to this kind of access event, and have this behavior override the default one. Of course, the capability of characterizing any access event and of associating any needed behavior to it makes it also possible to implement any form of event-notification mechanisms.

For instance, one can consider the enhancement of the basic tuple-space model towards a *programmable tuple space model* [DenNO98]. In this case, one must enable, for each and every specific (classes of) access to the tuple space – i.e., requests for storing, reading, or extracting given tuples performed by given classes of agents – to associate a behavior different from the default one, typically based on a built-in pattern-matching mechanism.

3.3 The Scenario of Context-Dependent Coordination

The adoption of local and programmable coordination media leads to strong and highly manageable forms of *context-dependent coordination*, in which coordination media are no longer only the mean via which coordination activities are enabled, but also the mean via which coordination activities are ruled.

On the one hand, coordination media associated to different execution environments may be independently programmed and, thus, may exhibit different behaviors in response to the same interaction events. This enables to integrate in the form of new reactions whatever needed *environment-specific coordination laws* to rule the local coordination activities, i.e., the ones of the *execution environment context*, transparently to agents and without requiring the introduction of specialized middleware components and/or of special-purpose agents.

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, behaviors. Thus, agents can carry the (mobile) code needed to implement and control *application-specific coordination laws*, and automatically install it in the form of new reactions into the coordination media of the visited sites. In this way, agents can coordinate on each site being guaranteed that the local coordination medium will let them coordinate accordingly to the needs of their specific *application context*, and without having to explicitly deal in their code with the coordination issues. This leads to a separation of concerns between algorithmic and coordination issues, the latter ones being dealt by computation embodied in the coordination media and dynamically distributed by agents in the sites they visit.

The scenario of context-dependent coordination is depicted in Figure 1. Agents interact with

local resources and other local agents always with the same, uniform, interface, via the API provided by coordination media. On a site, agents are subject to the local environment-specific laws, which can be different from environment to environment. In addition, the agents of a given application can spread their own coordination laws on the sites they visit, so that all the agents of the same application, even if executing on different sites, will have their coordination activities influenced by the same application-specific coordination laws (in addition to the environment-specific coordination laws). Of course, the coordination laws of a specific application, on any site, have to influence the coordination activities of only those agents belonging to the same application.

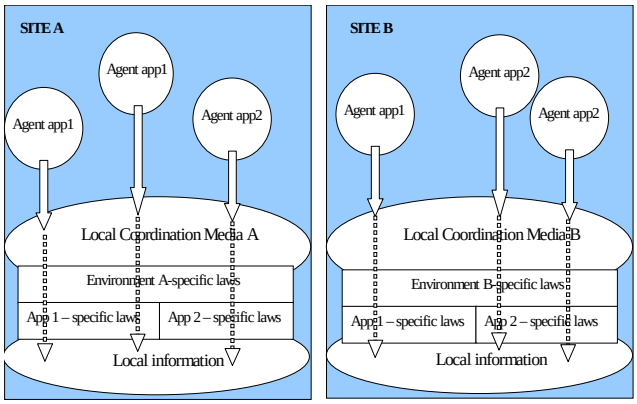

Figure 1. The scenario of context-dependent coordination.

3.4 Context-Dependent Coordination in MARS

To show how a programmable infrastructure for context-dependent coordination could look like, and how it can be used, this Subsection describes *MARS* (*M*obile *A*gent *R*eactive *S*paces), developed at the University of Modena and Reggio Emilia using Java, and described in an earlier prototype version in [CabLZ00b].

3.4.1 The MARS Architecture

Globally, the MARS architecture is made up of a multiplicity of independent programmable tuple spaces, associated to a node or to a domain of nodes, and acting as local coordination medium for that node/domain (Figure 2).

Although the first version of MARS supported only actually mobile agents, the prototype has been recently extended to support also virtually and physically mobile agents. Whatever the type of mobility, agents are supposed to access to a MARS tuple space via a private reference, which is bound to the MARS tuple space associated to the current execution environment of the agent. The

binding of the private reference to an actual tuple space changes accordingly to the agents' movement, and the only perceivable distinction between the different types of mobility relates to how the re-binding takes place.

With regard to virtual mobility, an agent can connect to a remote MARS tuple space (i.e., can virtually move to that context) via a special operation of "virtual migration" (see code in Figure 3) that provides for re-binding the private reference to the remote MARS tuple space to which the agent has virtually migrated, by exploiting Java RMI. It is worth outlining that agents can also disregard the "virtual migration" operation and can directly exploit the Java RMI mechanisms for connecting to a remote MARS tuple space and access it. This characteristic is important for enabling a MARS tuple space to be published as a Jini service and to be accessed accordingly.

With regard to actual mobility, when an agent migrates to a site (a single node or a node of a domain sharing a single MARS tuple space), the local MARS reference is automatically bound to the local MARS tuple space. In other words, the automatic side effect of the invocation of an "actual migration" operation is to update the binding of the MARS reference to the local MARS tuple space of the node to which the agent has actually migrated.

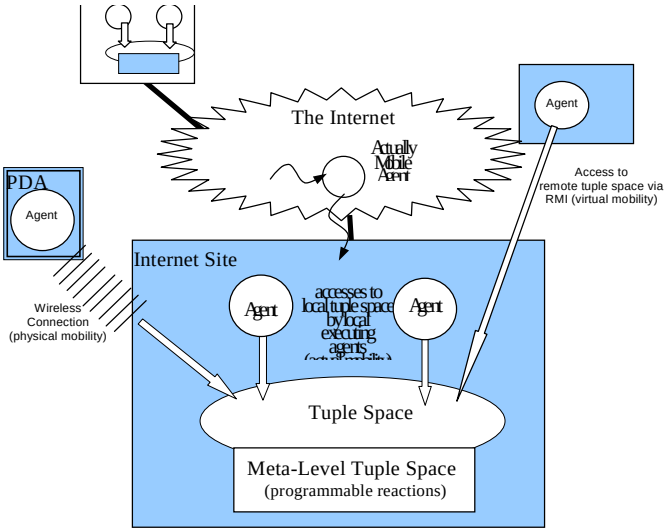

Figure 2. The MARS architecture.

With regard to physical mobility the implementation relies on a software agent running on a mobile computing device that is able to catch "connection events", i.e., those events generated by the wireless-enabled nodes of the fixed network infrastructure as soon as a device enters its connection range. Then, the solution adopted in MARS is to deliver the reference to the local MARS tuple space to a connecting device together with the connection event itself. The agent, by its side, can handle

this event at its willing, typically by binding its private MARS reference to the MARS tuple space to which it has connected. The current MARS implementation supports infrared and IEEE-802.11 technologies, and we are working towards the support for Bluetooth.

3.4.2 Using MARS Tuple Spaces

The MARS interface to access the tuple space is compliant with the Sun's *JavaSpaces* one [FreHA99]. Tuples and template tuples (both usually called "Entries") are Java objects whose instance variables, possibly with non-defined values in the case of template tuples, represent the tuple fields.

To access the tuple space, the write, read, and take operations are provided to store, read, and extract, respectively, a matching tuple. In addition, the readAll and takeAll operations are provided to read and extract, respectively, all matching tuples from the space. The default behavior of a MARS tuple space in response to access events is a quite traditional pattern-matching access to tuples of the tuple space. A tuple *T* matches a template tuple *U* (which has to be provided as parameter of read, take readAll and takeAll operations) if the defined values of *U* are equal to the corresponding ones in *T*. Access to a tuple space is controlled by means of Access Control Lists, establishing which agents can do which operation on which tuples (this mechanism can be complemented by a specific programming of the tuple space, as shown in Subsection 4.1.1).

With regard to the case study, Figure 3 shows the code fragment of a searcher agent roaming the Internet to retrieve information about HTML files. If the administrator of a site has put in the MARS space,ù tuples in the form: (filename, extension, keyword, modification_time, size), an agent migrated to that site can retrieve information about all HTML files related to the argument "coordination" by asking for tuples matching (string?, "html", "coordination", date?, int?) via a readAll operation.

3.4.3 Programming MARS Tuple Spaces

MARS enables the association of any needed reactions in response to access events performed by agents. This association occurs via *4-ples* characterizing the access event of interest, and stored in a "*meta-level*" tuple space. A meta-level 4-ple has the form of *(Rct, I, Op, T)*: it means that the reaction method (coding the reaction itself) of an instance of the class *Rct* has to be triggered when an agent with identity *I* invokes the operation *Op* on a tuple/template *T*. If a meta-level 4-ple has some non-defined values it associates the specified reaction to all the access events that match it. 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. Thus,

programming a MARS tuple space by installing/uninstalling reactions simply amounts at putting and extracting tuples from the meta-level tuple space. The administrator can do that via special-purpose agents or via a simple GUI. Application agents can do that via event-handlers [LanO98] that install the specified reactions automatically, without interfering with the agents' activities, and automatically downloading the required *Rct* class code if not locally available.

```
// declaration of FileTuple class
class FileTuple extends AbstractEntry  // tuple class
{ // tuple fields
  public String filename;  public String extension; public String keyword;
  public Date modification_time; public Integer size;
  // a proper constructor of the tuple can be also be defined
  // to initialize the tuple fields in a compact way
} // end of class FileTuple

class SearcherAgent extends Agent {
private Space LocalMARSSpace;
private FileTuple filePattern; private Entry[ ] HTMLFiles;
public void run()
{ …
   // the agent actually migrates to a site
  this.go_to("interesting.site.com");
  // this could also be a virtual movement such as
  //   this.virtual_go("interesting.site.com");

  // creation of template tuple of class FileTuple
  filePattern = new FileTuple(null, "html", "coordination", null, null);

  // retrieve from the space (via a readAll operation) the tuples matching the template
  // and representing HTML files related to the keyword "coordination"
  HTMLFiles = LocalMARSSpace.readAll(filePattern, null, NO_WAIT);
  // NOTE: the last parameter of the readAll operation represents
  // the time the agent has to wait for tuples, in the case no matching
  // tuple is immediately available.
  // the middle parameter has been added for compliance with the
  // JavaSpaces specification, but is not currently used in MARS
…}
```

Figure 3. An agent accessing a tuple space to retrieve information on HTML files (fragment).

When a reaction is executed, it is provided with *I*, *Op* and *T* as parameters, which can be used to drive the execution of the reaction itself. A vector of tuples (called InputTuples, and possibly with a single element) is passed as an additional parameter. This vector contains either *(i)* in the case of an input operation, the tuple(s) that would have been returned to the invoking agent as a result of the default pattern-matching mechanism or *(ii)* in the case of a write operation, the tuple that would have been written by default in the space. A reaction is expected to operate on that vector and to return it as a result, determining either the tuples to be returned to agents or the tuple to be written in the space. In this way, a reaction is allowed not only to fully override the default behavior of the tuple space, but also to simply increment or slightly modify it. It is worth noting that a reaction, in its code, can perform any needed access to the base-level tuple space. However, to avoid endlessly recursive reactions, the accesses to the base-level tuple space performed inside a reaction do not trigger further

reactions.

Figure 4 shows a simple reaction class devoted to count the number of accesses to a tuple space. By inserting, for example, the 4-ple (CountReaction, null, null, null) in the meta-level tuple space, the reaction gets associated to all operations, whatever the identity of the agent performing it and whatever the associated template tuple.

To ensure consistency in the execution of a reaction, the involved tuples (e.g., the tuples in the InputTuples vector) are locked during the execution of a reaction. However, other reactions (as well operations with a standard behavior) can execute concurrently if they involve different tuples. This guarantees a high-degree of concurrency in the activities of the agents accessing to the tuple space.

```
class CountReaction implements Reactivity
{ private int totalAccesses = 0;
  // number of accesses performed by agents

public Entry[ ] reaction(Space s, Identity Id, Operation Op, Entry Template, Entry InputTuples [ ])
{           totalAccesses++;  // increment counter
            return InputTuples; // return result tuples without change }
} // end class CountReaction
```

Figure 4: A simple reaction counting the number of accesses.

3.4.4 Composing MARS Reactions

When, in response to an interaction event, several 4-ples (i.e., several reactions) satisfy the meta-level pattern-matching mechanism, all the corresponding reactions are executed in a pipeline. In this case, the InputTuples vector passed to a reaction contains either the tuples returned from the execution of the previous reaction in the pipeline, if any, or the tuples resulting from the standard pattern matching, otherwise. At the end of the pipeline, the system actually performs the operation (i.e., returns one or all tuples to the agent, deleting them if necessary, or writes the tuple in the space, depending on the kind of operation). The global behavior of an operation results from the composition of the behavior of each of the reactions in the pipeline. Again, for the sake of consistency, the tuples in the InputTuples vector are locked for the whole execution of the pipeline.

The fact that a reaction can be associated to access events performed by agents with a specific identity as well as independently of the identity of the accessing agents, and the fact that reactions can be composed with each other, enables for both environment-dependent coordination and application-dependent behaviors to be installed in a MARS tuple space. On the one hand, depending on its own needs, the administrator can install reactions that apply to all the agents executing on the node, or to specific classes of agents. On the other hand, agents can install their own application-specific reactions on the visited sites. The security mechanisms of MARS can control – via Access

Control Lists on the meta-level tuple space – which classes of agents can associate reactions to which access events, and can thus also guarantee that an application-specific reaction keeps confined to a specific application context.

By default, the order of the execution of the reactions in a pipeline is determined by their installation order. However, the administrator can override the default by specifying where in the pipeline reactions have to execute. This gives the administrator full control over the global behavior of pipelined reactions and over possible inconsistencies.

4 APPLICATIONS OF CONTEXT-DEPENDENT COORDINATION

In this section, we describe several classes of problems for which context-dependent coordination – both environment-dependent and application-dependent ones – enforced via programmable coordination media, can be effectively exploited in Internet-agent applications. The case study introduced in Subsection 2.3 is extended and specialized to exemplify the expressed concepts and to show their actual implementation in MARS.

4.1 Environment-Dependent Coordination

Agents, while accessing a local coordination medium always with the same interface, can have their coordination activities affected by the specific behavior programmed in the local space. This can be used both to embed access control policies in the local space and to help agents entering a foreign environment, accessing local resources and interacting with local agents, without having to explicitly take care of all the issues implied in that.

One can criticize that programming the coordination medium and changing the effects of the performed operation transparently to agents can be dangerous and can produce unpredictable and/or uncontrollable effects on applications. However, we emphasize that the change due to a specific environment-dependent programming is observable only from the internal of the coordination medium. From the point of view of application agents, a specific programming of a coordination medium simply results in a different perception of its content and state, e.g., in the case of a tuple space, in a different view of the tuples actually contained.

4.1.1 Enforcing Access Control Policies

When a site opens itself to the execution of mobile agents, it must protect its resources from unauthorized accesses performed by malicious or badly programmed agents, or by agents that are

simply unaware of their local permissions. A programmable coordination medium can be used to support and complement standard security mechanisms – typically based on static access control lists – with dynamic security and resource control policies expressed in terms of specific behaviors to be assumed by the coordination medium in response to access events. The possibility of expressing dynamic security and resource control policies can also be used to facilitate the execution of those agents that are unaware of the local policies, and for which a large number of security exceptions are likely to be thrown. In fact, the local coordination medium can be programmed so as to make any unauthorized access harmless without the need of generating any exception.

```
class LimitNumber implements Reactivity
{  // number of maximum tuples to be read by an agent
   private int maxReadings = 100;

  public Entry[ ] reaction(Space s, Identity Id, Operation Op, Entry Template, Entry InputTuples [ ])
  { ReadingsTuple pastReadings;  // accounting for previous readings

    int alreadyRead = 0; // number of tuples already read by this agent
    // try to read an accounting tuple for this specific agent
    pastReadings = (ReadingsTuple)s.read(new ReadingsTuple(Id, null), null, NO_WAIT);

    // if a tuple is found, this agent has already read local tuples
    if (pastReadings != null)
            alreadyRead = pastReadings.alreadyRead.intValue();
    if (alreadyRead == maxReadings) return null;
    // no tuple to be returned
    // else it can read tuples and a result array
    // where to copy the remaining number of tuples is created
    int dim = min(result.length, (maxReadings – alreadyRead));
    Entry result[ ] = new Entry[dim];
    // copy the allowed number of tuples
    System.arraycopy(InputTuples, 0, result, 0, (maxReadings - alreadyRead));
    // store the new information  about the new number of readings
    s.write( new ReadingTuple(Id, new Integer(alreadyRead + result.length )));
    return result;
  } // end of method reaction
} // end of class LimitNumber
```

Figure 5. A reaction that constrains the number of tuples retrieved.

The above considerations apply to any application area on the Internet. As a specific example applied to the case study, a tuple space can be programmed so as to let a searcher agents in charge of retrieving information about HTML files (i.e., the agent in Figure 3) read only a specified maximum amount of HTML tuples, for which is ensured the charge coverage from the related user. If an agent tries to retrieve a tuple once it has already retrieved the specified amount, the tuple space, depending on the chosen policy, can either *(i)* generate an exception or *(ii)* simply appear to the agent as if it does not contain further HTML tuples. In the latter case, the agents do not experience any exception and simply perceive a different view of the tuple space content, while the environment can still enforce its access control policies.

The code of a MARS reaction implementing such policy is reported in Figure 5. The reaction can

be installed by the administrator by writing a 4-ple (LimitNumber, null, readAll, (Filetuple)(null, “html”, null, null, null)) in the meta-level tuple space, to indicate that the reaction of the class LimitNumber has to be triggered when any agent requesting an HTML tuple via a readAll.

4.1.2 Handling Heterogeneity

Different execution environments can adopt different choices with regard to the representation of the data and resources accessible via the local coordination medium. Agents, in these cases, have to explicitly deal with heterogeneity and somehow discover how to fruitfully access the environment. When adopting a programmable coordination medium, a completely different perspective can be adopted. In particular, the coordination medium can be programmed so as to react to the accesses performed by agents by transforming the agents’ requests in accord to the local representation. Therefore, on the one hand an agent can perceive the local data and resources as if it were homogeneous to its expectations; on the other hand, the environment is not forced to change its local representation of the environment’s resources or duplicate it in different formats and/or in different middleware components.

Shifting the burden of handling heterogeneity from agents to environments may be criticized, in that it assumes that the environment (i.e., its administrator) knows in advance all possible types of agents that will attempts to access the coordination medium. This is not the case. Simply, an administrator can select those known agents’ type of interest (e.g., those of widespread applications) and tune the coordination medium to them, letting other agents’ types in charge of autonomously discover how to fruitfully access the coordination medium. An additional reason for handling heterogeneity via a programming of the coordination medium relates to fact that, more and more, the Internet is exploited as a collaboration platform for dynamically and temporarily formed teams of people and enterprises [JaiAS99]. These workgroups/enterprises, for specific and limited amount of time, are in need of collaborating and sharing portions of their knowledge and services with each other, to form a temporarily common infrastructure. Programmability can then be used to make the data and services of each member of the group easily available to the other members, without forcing a strong re-engineering of the local information systems.

Let us consider a very simple example related to handling heterogeneity in the case study. 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 of HTML pages in 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 for "html" files into requests for "htm" files, transparently to agents. In other words, the administrator modifies the default behavior so as to make "html" match with "htm".

With reference to MARS, such a reaction is shown in Figure 6, It can be installed on a site by writing the 4-ple: (HTML2HTM, null, readAll, (FileTuple)(null, "html", null, null, null)) to express that the reaction has to be triggered when an agent invokes a read operation with the template (null, "html", null, null, null). This reaction can be associated also to read operations without any change.

```
class HTML2HTM implements Reactivity
{
  public Entry[ ] reaction(Space s, Identity Id, Operation Op, Entry Template, Entry InputTuples [ ])
  // no match already occurred if the site has only "htm" files
  { // modifies the extension of the required files
    ((FileTuple)Template).Extension = "htm";
    // require matching with new extension
    Entry [ ] result = s.Op(Template, null, NO_WAIT);

    for (int i=0; i < result.length; i++)
            // change back the extensions in the found tuples
            ((FileTuple)result[i]).Extension = "html";
    return result;
  }  // end of method reaction
} // end of class HTML2HTM
```

Figure 6. A reaction that makes HTML match with HTM.

4.1.3 Supporting Open Interactions

Exploiting the programmability of the coordination medium 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, possibly heterogeneous in terms of supported languages and protocols, and nevertheless in need of coordinating with each other. A programmable coordination medium can be used as a mediator, so as to support the coordination activities in a group of heterogeneous agents. First, it can be exploited to provide the needed syntactic and semantics translations to enable, transparently to agents, fruitful conversations. Second, it can be exploited as a repository of the "rules" according to which the agents' activities have to be coordinated. For example, in the case of agents acting on behalf of a workgroup or of a virtual enterprise, such rules can determine which documents and data sources can be read/modified at a given time by which agents [Tol00].

More generally, a trusted site can be used to act as an impartial arbiter, in charge of controlling the interactions of agents with other agents, possibly self-interested, interactions that would be likely to

be unfruitful or damaging otherwise. The most notable examples in which this way of exploiting a programmable coordination medium can be useful are agent marketplaces. In the near future, agents are expected to roam the Internet to join marketplaces – typically, specific Internet sites – and there negotiate for the acquisition of goods and resources. This trend is made evident by increasing diffusion of Internet auctions and by the recognized advantages of mobile agents in this context [SanH00]. Any marketplace, as usual, has its own rules, and it requires that the agents on a marketplace there negotiate accordingly to the local rules. For instance, any specific auction defines rules concerning the way to make bid, the way goods are put on sale, and the way the winner and the final price are eventually identified. In that scenario, the adoption of a context-dependent coordination model is a very natural choice. Environment-dependent coordination can be exploited to let the coordination medium control that the negotiation activities of agents go on accordingly to the local negotiation rules. In the case of an auction, the environment can control the correct timing of auction process, the correctness of submitted bids, the possible emergence of collusion among bidders and/or among bidders and sellers, and so on, depending of the specific negotiation rules that have to be locally imposed.

As an example related to the case study, let us suppose that the searcher agents, after having retrieved information on the resources of a file, decides to "buy" the access to some digital resource, e.g., a gif image, that it has discovered being actually sold. To this end, the agent can put a "bid" tuple in the local MARS tuple space to express its willingness to buy the access to that file at specific conditions (see Figure 7). Then, before starting accessing the file, the agent waits for reading an "ack" tuple informing it of the success/failure of the transactions. This implies the presence of a local seller agent, in charge of performing the necessary commercial transactions, of providing access to the file, and eventually of notifying the buyer agent via the writing of the "ack" tuple. However, in the presence of a programmable tuple space, one can use the programmability to perform some preliminary control on the submitted bids, to check if they are correct or, instead, if they are not in accord with the local rules and, as that, can be refused immediately without any involvement of the seller agent. The code implementing such a behavior is presented in Figure 7, and it can be installed in a MARS tuple space by writing the meta-level tuple: (CheckBid,null,write, (BidTuple)(null,null,null)).

4.2 Application-Dependent Coordination

Application designers can exploit the programmability of the coordination media in different

ways, to adapt the interaction model to their specific application-dependent needs or, more generally, to embed specific coordination laws into the coordination media. It can be used to facilitate the access to the information on the visited site via a customization of the coordination medium, to support implicit forms of coordination between agents, and to implement and control complex communication protocols. Of course, since these modifications of the coordination media' behaviors are intended to meet specific application needs, some forms of confinement, as the one integrated in MARS, must be provided to ensure that any application-specific behavior will affect only the access events performed by agents of that specific application.

```
// Excerpt of the agent code requesting to buy a resource
…
// creation of a bid tuple
bidt = new BidTuple(ChosenGifFile, MyBid, MyIdentity);
// write a tuple requesting to buy the chosen gif image
// at a price and at the conditions described by MyBid
s.write(bidt, null, 1000);
// NOTE: the second parameter is not used in MARS; the third parameter is the time to leave of the tuple in the space

// template tuple for reading the answer tuple
ackt=new AckTuple(ChosenGifFile,MyIdentity, (Answer)null);
// read the answer tuple
acka = s.read(ackt, null, 1000);
if(acka.answer == true) // transaction successful
        {…// go on with the process and access the file
        int k = ChosenGifFile.dim;
        // etc etc…
…
// code of the reaction performing a preliminary control on the correcteness of the submitted bid
class CheckBid implements Reactivity
{ AckTuple ackt;
  public Entry[ ] reaction(Space s, Identity Id, Operation Op, Entry Template, Entry InputTuples [ ])
  { //check the correctness of the bid, for example:
    // the bid price should be greater or equal than the price for
    // accessing the specified resource

if((BidTuple)Template.bid.price<(BidTuple)Template.file.getPrice())
    {// the price is less than required, the bid is not valid
    // create and write ack tuple to be read by the buyer agent
    // and notifying it of the failed transaction
    ackt = new AckTuple(..);
    ackt.answer = false;
    s.write(ackt, null, 1000);
    // return null, which imply that that the bid tuple
    // will not be written in the space
    return null;  }
 else
 {  // if the bid is valid
    // return the bid tuple, which will be written in the space
    // and will be eventually read by the seller agent
    return Template;}
  } // end of method reaction
} // end of class CheckBid
```

Figure 7. Code of the agents submitting a bid to buy access to a resource, and code of the reaction controlling the correctness of the bid.

The programmability of coordination media can sometimes make agents lighter and less expensive to migrate – the code implementing the coordination laws may be found in the visited sites

without requiring remote downloading – and can make it possible to write less code – the code for programming the coordination medium to enact specific coordination laws can be reused in different applications. However, the very advantage is to promote modularity via a clear separation of concerns between computational issues and coordination ones.

4.2.1 Customizing the Coordination medium

A proper programming of coordination media can be exploited by application agents to effectively access the resources of a site via a customization of the access mechanisms. In fact, the default behavior of a coordination medium cannot always fit all the needs of all possible applications. This is testified by several proposals for extensible server architectures, enabling the download of mobile code to servers with the aim of customization [StaG90, SUN97]. A programmable coordination medium can naturally provide such feature, by enabling agents to install any needed access service in the visited coordination media.

```
class MyMatch implements Reactivity
{
  public Entry[ ] reaction(Space s, Identity Id, Operation Op, Entry Template, Entry InputTuples [ ])
  { if (((FileTuple)Template).Keyword.indexOf('*') < 0)
    // if the keyword does not contain a wildcard
            return InputTuples[ ];
     else // no match has occurred if the keyword contains a wildcard "*"
     {
        FileTuple resultTMP[ ], result[ ]; int cont = 0;
       // a template tuple with null as Keyword => retrieve all keywords
       FileTuple tmp = (FileTuple)T.clone(); tmp.Keyword = null;
       // read all potential matching tuples
       Entry [ ] readTuples = s.readAll(tmp, null, Op.timeout);
       for (int i = 0; i < readTuples.length; i++) // for each retrieved tuple
         {  // if the keyword field does not (My)match
            // keyword field of the template
            if(!this.keywordMatching((FileTuple)readTuples[i]).Keyword, ((FileTuple)T).Keyword))
              readTuple[i] = null; // delete it from the result array   }
         // return the array without null elements
         return this.compactArray(readTuples);
    } // end of else
  }  // end of method reaction
} // end of class MyMatch
```

Figure 8. A reaction changing the built-in pattern matching.

Let us exemplify the above aspects by directly referring to the case study. It could be useful for searcher agents to specify the Keyword field of a FileTuple tuple in a UNIX style, i.e., by exploiting wildcards such as '*'. This extension would grant more flexibility in information retrieval. However, the basic pattern matching mechanism forces a searcher agent to firstly retrieve all the tuples representing files and, only subsequently, to select those having the Keyword field matching the given string in a UNIX wildcard style. Therefore, this application calls for a different pattern-matching mechanism, which can enable the selection of all HTML tuples in which the Keyword field UNIX-like matches with the expression specified in the template.

Figure 8 shows how this pattern-matching can be implemented in MARS by the class MyMatch. It can be associated to the readAll operations performed by agents with the specific_app_id application identifier via the 4-ple (MyMatch, specific_app_id, readAll, (FileTuple)(null, "html", null, null, null)).

4.2.2 Enabling Implicit Coordination

The programmability of coordination media can be used by application agents to enable peculiar forms of implicit coordination between agents. Instead of having agents issue specific coordination actions, one can have the coordination media themselves direct the agents' actions so as to achieve a needed coordination pattern. In particular, agents can program the visited coordination media so as to make them affect – via a peculiar internal behavior and via the produced information – the coordination activities of the other application agents to be involved in a coordination pattern. This can be useful whenever the application evolution may require agents to coordinate their activities and/or to exchange information, and either the agents are unaware of this coordination need or the designer has decided to let agents disregard it, because too complicated to be handled by agents themselves.

In the case study, to avoid duplicated work, we have previously suggested exploiting marker tuples, to be put by the agents into the tuple spaces of the visited sites, and to be looked for by agents arriving on a site. This solution defines an explicit coordination among the agents of the application that is likely to complicate the agent design. Also, it requires an agent to know *a priori* that it has to look for some information and that it will be possibly required to coordinate its activity on a site. In the presence of a programmable tuple space, 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 added or modified since the last visit. In this 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.

A reaction that implements the above behavior in MARS is shown in Figure 9, and it can be installed by having agents write the 4-ple (ModifiedOnly, specific_app_id, readAll, (FileTuple)(null, "html", null, null, null)) in the meta-level tuple space. To have a similar reaction work with read operations, one could simply keep track of the retrieval time of each tuple.

4.2.3 Implementing Communication Protocols

Whenever two agents have to interact accordingly to specific communication protocols, and in the absence of a programmable coordination medium, agents are typically in charge of integrating in

their code the capability of handling and controlling the proper execution of these protocols. This is not generally a problem in object-based applications, where communication protocols typically follows simple request-reply patterns. In agent-based applications, instead, the execution of an application often involve complex, multiphase and stateful, communication protocols between its agents.

```
class ModifiedOnly implements Reactivity
{ private Date visit;  // date of the last visit

public ModifiedOnly()   {visit = new Date(); }
// the constructor sets the date of last visit

  public Entry[ ] reaction(Space s, Identity Id, Operation Op, Entry Template, Entry InputTuples [ ])
  { // for each matching tuple (if any)
    for (int i = 0; (InputTuples != null) && (i < InputTuples.length); i++)
        // if this document was NOT modified since last visit
        if (((FileTuple) InputTuples[i]).ModificationTime.before(visit))
            InputTuples[i] = null; // it is not returned to the agent
    visit = new Date();   // set the time of the last visit
    // return only the updated tuples (null elements are purged)
    return this.compactArray(InputTuples);
  } // end of method reaction
} // end of class ModifiedOnly
```

Figure 9. A reaction that returns modified files only.

For instance, agents in charge of performing a specific task in an application are often designed, by exploiting the agents' autonomy, so as to negotiate the provision of services and information, as well as the assignment of tasks to be performed [Smi80]. This mimicking of real-world interaction models can be very useful to achieve a wise exploitation of resources and a wise distribution of tasks, as it happens in real-world organizations of autonomous entities. Moreover, the need for agents to negotiate and exchange knowledge accordingly to complex protocols can often derive from the nature of the application scenario, when agents execute as entities delegated to perform transactions and negotiate in commercial or industrial contexts.

When such complex communication protocols are to be integrated in an application, a programmable coordination medium can be exploited – via a proper programming – so as to keep track of the state of execution of the protocol, react to access events accordingly, and enforce a correct execution of the protocol. In this way, agents can be freed from the duty of controlling protocols: the required code is no longer part of the agents, but it is simply something that agents dynamically and transparently install in the coordination media of the visited sites.

When talking about agent communication protocols, one cannot forget that, in most of available agent systems, agents interact in a conversational way accordingly to an Agent Communication Language (ACL): messages exchanged between agents represent structured speech acts of a

conversation [Fin94, FIPA]. Although we have not yet directly faced this issue, our proposal for tuple-based coordination media can naturally be extended to support conversational, ACL-based, model of interaction between agents, e.g., the standard FIPA ACL. In fact, the structured messages of an ACL can be naturally mapped into tuples. For example, an ACL message like:

```
(request
  :sender AgentA
  :receiver AgentB
  :ontology fipa-agent-management
  :language SL0
  :protocol fipa-request
  :content (...)
)
```

can be mapped into the following tuple: (String kind_of_speech_act, String sender, String receiver, String ontology, String language, String protocol, Object content). Putting together the above mapping possibility with the programmability of the media, one can easily see that any conversation between two agents can take place via controlled exchanges of ACL-like tuples through a coordination medium.

As an example, the OneToMany reaction in Figure 10 can be used to realize a one-to-many request, by associating it to a writing of a tuple representing an ACL message. When AgentA sends a request message (writing an ACL-like tuple) to AgentB, the reaction writes an additional request tuple in the space, so as to send the request also to a further agent, AgentC. This can be useful in applications where new kinds of agents are added dynamically, and they must be enabled to be involved in communication protocols.

```
class OneToMany implements Reactivity
{
public Entry[ ] reaction(Space s, Identity Id, Operation Op, Entry Template, Entry InputTuples [ ])
  { // the tuples to be written represents an ACL message
    ACLMessageTuple t = (ACLMessageTuple)Template;
   // if the receiver agent is AgentB
    if (t.receiver.equals("AgentB"))
    {  ACLMessageTuple nt = t.clone();// create a new tuple and set the receiver
      nt.receiver = "AgentC";
      // send the message also to AgentC in the form of tuple
      s.write(nt, new Transaction(null), 10000);
    }
    // anyway, the message tuple is returned to be written
    return Template;
    } // end of method reaction
} // end of class OneToMany
```

Figure 10. A reaction that delivers a message to the receiver agent and to another agent.

5 IMPACT ON DESIGN AND DEVELOPMENT

The adoption of context-dependent coordination concepts and the availability of an infrastructure based on programmable local coordination media can have a very positive impact on the engineering

of mobile agent applications.

5.1 Impact on Design

From the point of view of application designers, context-dependent coordination represents a useful conceptual framework. In fact, it naturally invites in designing an application by clearly separating the *intra-agent* issues – related to the specific computational tasks in charge to 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, context-dependent coordination promotes a clear separation of concerns, which is likely to reduce the complexity of application design (see Table 1).

	Design Phase	**Development Phase**	**Case Study**
Intra-agent Issues	Identification and specification of agents' tasks.	Code of the agent in charge of performing specific tasks.	All agents have to achieve the task of retrieving information from Internet sites. Have to migrate in the Internet, clone themselves, access to local information and extract information.
Inter-agent Issues	Identification of needed interactions between agents and between agents and environments. Identification and specification of global rules for interactions.	Code implementing the application-specific coordination laws, to be installed at run-time by agents.	Agents have to avoid duplicated work on a site.
Site Administrator Issues	Identification of environment-specific issues related to agents' accesses.	Code implementing the environment-specific coordination laws.	The local HTML information should be made easily accessible in a safe and secure way to agents.

Table 1. Separation of concerns enforced by context-dependent coordination.

Without the ambition of introducing a detailed methodology, we can sketch the general guidelines for the identification of intra-agent and inter-agent issues [ZamJW01]. At the intra-agent level, the designer has to firstly analyze the global application goal and decompose it into sub-goals, i.e., sub-tasks. Then, different sub-tasks have to be assigned to different agent classes, whose instances will attempt to achieve in autonomy. At the inter-agent level, the designer has to identify, for each of the tasks to be achieved by agents, what kinds of interactions with other agents and with the local execution environments are necessary for each agent to succeed in achieving its assigned sub-tasks. This also implies identifying which rules must be enacted on interactions for the global application goal to be effectively and coherently achieved.

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

- *intra-agent design*: all of the agents in the application are assigned the same task of searching information in the Web, although different agents are in charge of a sub-portion of the search space. Defining that sub-tasks implies identifying: what kind of information the agents should retrieve on the visited sites; how they should analyze this information and extract useful data from it; how they should report data back to the user; how they should select their itinerary and possibly clone themselves.
- *inter-agent design*: agents access to local data and different agents may visit the same site several times. These problems may require identifying whether agents require specific mechanisms and solutions to access to local data, and how agents can avoid duplicated work on a site to achieve the global application goal in an efficient way, respectively.

The analysis of the inter-agent engineering issues leads to the definition of the application-specific coordination laws to be spread into the visited sites by application agents, which are independent of the design choices deriving from the analysis of the intra-agent engineering issues. For instance, no matter what the method for information extraction is integrated into the agents of the case study, 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 for having agents retrieve only those HTML pages that have been modified since the last visit.

Independent of the duty of application designers is the duty assigned by context-dependent coordination to *site administrators*. When new kinds of application agents are known to be going deployed on the Internet (or when sites are going to federate to share a common infrastructures) the administrator of one site can identify all the environment-specific coordination laws that she may find it necessary to facilitate the execution of the agents on the site, to made it homogeneous to agents' expectations (or to the requirements of the forming federation), and to protect the site from improper exploitations of the local coordination medium.

In the case study, as already discussed, administrators can decide to limit the amount of tuples read by application agents, and to handle tuple extraction attempts without issuing exceptions in agents. Also, they can dynamically provide different representations of HTML tuples, to let agents access tuples event if their requests do not match the actual representation.

5.2 Impact on Development

The separation of concerns of the design phase can be preserved during the development and maintenance phases too (see Table 1). Given the availability of a programmable coordination infrastructure, the code of the agents can be clearly separated from the code implementing the coordination laws (whether environment- or application-specific ones): agents and coordination laws can be coded, changed, and re-used, independently of each other. Thus, by avoiding to hardwire into agents the code related to the implementation of specific coordination laws, context-dependent coordination promotes the writing of more modular, manageable, and re-usable code.

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 at a site retrieve only those HTML pages that have been modified since the last visit. Later in time, the designers and/or the developers can discover that different coordination laws may be needed to achieve effective information analysis. For instance, they can discover that an agent arriving at a site should analyze – in addition to those HTML pages that have been modified since the last visit of another application agents – also all those unmodified local pages that are linked to the modified ones. Since the coordination laws are clearly separated from the code of the agents, the code of these new laws can be made easily and dynamically take place in the application, by simply having the unchanged agents distribute over the visited site the code of the updated coordination laws. Thus, not only the updating can take place without having to rebuild the agents, but it can take place while the application is in execution. In a decentralized and "always-on" world, this is indeed an important characteristic. A similar approach can be followed by the administrator of one site, which can change at any time its environment-specific laws without forcing the agents that want to visit the site to adapt to the changes or to stop their activities.

In MARS, the clear separation between the code implementing the agents and the one implementing the reactions makes the above advantages on code development and maintenance immediately applicable. Moreover, since MARS reactions can be combined in a pipeline, it is possible to install reactions on a tuple space, uninstall them, or change the code and behavior of a

reaction without changing the agent and/or the other reactions. For instance, the reactions of Figure 5 and Figure 6 can be combined in a harmless way: the tuple space can react with an HTML2HTM-LimitNumber pipeline to readAll operations requesting for "html" tuples; the HTML2HTM returns all "htm" tuples, which are used as the input of the LimitNumber reaction that may reduce the number of the tuples returned to the agent. Further, these reactions can be combined at any time with the application-specific ModifiedOnly reaction of Figure 9, which enable the retrieving of only the tuples updated since the last visit.

As anticipated in Subsection 3.4, the administrator has control over the order in which reaction are executed in the pipelines. Typically, the reactions to be triggered as first are the environment-specific ones in charge of dynamically producing the needed tuples in accord to the local coordination laws (e.g., in the case study, the HTML2HTM one), which can then be followed by dynamically installed application-specific reactions (e.g., ModifiedOnly the one). At the end of the pipeline, the administrator can place the execution of those environment-specific reactions devoted to implement specific security policies (e.g., the LimitNumber one) and, if it can be the case, to bound the effects of application-specific reactions.

The advantages of context-dependent coordination and of programmable coordination infrastructures make us encourage their adoption. However, we also want to emphasize that the presence of an infrastructure based on programmable coordination media does not necessarily force designer, developers, and administrators to commit to a context-dependent approach. They are free to neglect the programmability of the coordination media and develop applications in terms of agents interacting via non-programmed coordination media, to be exploited simply as the media via which interactions occur. Even more important, the enrichment of coordination media towards programmability does not undermine the usability of applications developed for non-programmable coordination media. For instance, applications developed to be executed relying an infrastructure based on the non-programmable JavaSpaces' tuple spaces [FreHA99] can normally execute relying on the programmable MARS infrastructure (which, we recall, is JavaSpaces compliant).

5.3 Performance Issues

A programmable coordination infrastructure promoting context-dependent coordination, while providing advantages during design and development, is likely not to significantly penalize applications from the performance point of view. Table 2 reports the times needed by an Aglets agent

[LanO98], running on a Sun ULTRA 10, to perform a read operation in a local MARS tuple space and return a matching tuple. Two different cases are analyzed: *(i)* with the meta-level activities off (*passive* case), as if MARS were a non-programmable tuple space implementation, and *(ii)* with the meta-level activities on, and by having the read operation performed by the agent issue one null-body reaction (*reactive* case). For all cases, as expected, the access times to the MARS tuple space increase nearly logarithmically with the number of tuples stored in the base-level tuple space (from 4 to 10 ms in the tests performed), due to the larger amount of information to be dealt with in the pattern-matching process. By comparing the access times to the tuple space with and without the meta-level matching mechanisms activated, one can see that the overhead introduced by the MARS reactive model is very limited, well below 15% independently of the global number of tuples in the base-level. Of course, in a real-world case, any reaction installed in the tuple space will have a non-null body, whose execution will make the tuple space access times increase. However, this cannot be considered as a performance drawback of the model, in that the same code executed in the reaction would have been executed by the agent, in the absence of context-dependent coordination. Similar performance considerations are expected to apply to other programmable architectures based on different models.

	# of tuples			
	100	**200**	**300**	**400**
passive case	3,9 ms	4,5 ms	4,9 ms	5,1 ms
reactive case	4,3 ms	5 ms	5,4 ms	5,7 ms

Table 2: Time (ms) to access to a MARS tuple space.

6 RELATED WORK

A few of the proposals in the area of Internet agents explicitly focus on coordination models suited to mobility, and, to our knowledge, no other proposals explicitly deal with context-dependent coordination concepts. However, several systems and models, in different areas, focus on problems (and propose solutions) strictly related to the ones discussed in this paper.

6.1 Coordination Models and Infrastructures

The *T Spaces* project at IBM [IBM98] and the *JavaSpaces* project at Sun [FreHA99] define Linda-like tuple spaces to be used as both general-purpose network information repositories and coordination media. However, they do not define infrastructures 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. In addition, neither T Spaces nor JavaSpaces define a programmable

coordination media. Nevertheless, T Spaces recognizes the needs for dynamically adaptable coordination media, and integrates a peculiar form of programmability, by enabling new complex primitives (i.e., queries) to be added to a tuple space. This 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.

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 directly via URLs, as an Internet service. As in MARS, this enforces network-awareness and virtual mobility without forcing actual agent mobility. 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 any kind of active mobile entities, and without requiring the presence of a fixed network infrastructure [PicMR99]. Each “agent”, whether a software agent, an Internet site, or a physical mobile device, owns and carries on in its movements 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 spaces. 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. A similar approach has been followed in the implementation of the Fishmarket systems for agent-mediated auctions [NorSR98], with the goal of guaranteeing that auctions can proceed accordingly to specific auctions' rules. Although not explicitly addressing agent mobility issues and application-dependent coordination issues, both the above approaches are suitable to enforce environment-specific coordination laws in systems of agents interacting in a peer-to-peer way.

As an additional note, it is worth outlining that context-dependent coordination is perfectly in accord with the current key concepts of mobile and ubiquitous computing [Wei93, AboM00], namely *context-awareness* and *invisibility*. The idea behind context-awareness is that the computation performed in mobile and ubiquitous settings has to be aware of and has to strongly depend on the context in which it occurs. The context of a computation is typically identified by *who* perform the computation, *what* the computation is intended for, *where* (i.e., in which physical environment) it occurs, and *when*. Such definition of context is very similar to the one exploited by the MARS programmable model, in which the effect of coordination activities may depend on the agent identities (*who*), on the operations performed (*what*), on the sites in which they take place (*where*) and on the current state of the media (*when*). The idea behind invisibility is that the presence of computations, despite pervasive in a given environment, must be invisible to users. In the same way, MARS reactions can support the agent's coordination activities without the need of being invoked or dealt with explicitly by agents.

6.2 Coordination in Multi-agent Systems

Several works in the area of multi-agent systems and of agent-based software engineering are recently recognizing that modeling and engineering interactions in complex and open multi-agent systems cannot simply rely on the agent capability of communicating via agent communication languages and of acting accordingly to the need/expectations of each agent in the system. Instead, concepts such "organizational rules" [Jen00, ZamJW01], "social laws" [MosT95], "social conventions" [NorSR98], "active environments" [Par01] are receiving more and more attention. Under different terminology and techniques, the common idea is that, for the effective engineering of multi-agent systems, higher-level, inter-agent, concepts and abstractions must be defined to explicitly model the environment (or the society) in which agents live and have to interact, and the associated environmental (or societal) laws.

As an example, several researches show that engineering a complex multi-agent system may take advantage of an “organizational approach”: agents are considered as entities that autonomously play a role in an agent-organization, with a set of goals (sub-tasks) to be achieved, and interacting (by collaborating or competing) with the other agents of the systems coherently with the role duties. This approach is promoted by the fact that, in several cases, multi-agent systems are deployed to support and automate the activities of some real-world organization, as well as by the fact that (as anticipated in Subsection 4.2.3) real-world interaction models can promote higher efficiency and adaptability. However, it is being recently recognized that, for the effective design and development of an organization of autonomous agents, the identification of the roles to be played by agents and of the inter-role patterns of interactions is not enough. Rather, defining an organization also requires the identification of the global constraints under which the interactions between roles in the organization have to occur. The identified global constraints can then be hardwired into the agents to limit their possible actions [MosT95], or they can be used to select the most suitable organizational structure for the system [ZamJW01]. The concept of context-dependent coordination is likely to facilitate the definition and the implementation of complex multi-agent systems on the Internet in accord with the above organizational perspective. On the one hand, concepts such as organizational rules can be easily mapped in terms of application-specific coordination laws. Agents can enforce such laws in a distributed way via a proper programming of the visited coordination media, without having to hardwire them into agents or into a specific organizational structure. On the other hand, an Internet site hosting the execution of mobile agents can be conceived as an open organization in which agents arrive to execute specific tasks. Thus, despite belonging to their original organization, agents also play a role in the foreign organization identified by the local site, which can enforce its own organizational rules via environment-specific coordination laws.

As another example, in ant-based systems complex global behaviors can emerge from the interactions of very simple agents with a computational environment [Par97, Par01]. Agents put and sense pheromones in the environment, and act accordingly to the concentration of pheromones in given sections of the environment. The environment, by its side, makes pheromones vanish with time, according to specific laws. The global behavior that emerges from the ant-colony strictly depends on the laws upon which pheromones vanish: in other words, the environment plays an active role by influencing the overall behavior of the system via an imposition of environment-specific rules. Again, the concept of programmable coordination media promoted by context-dependent

coordination represents the most natural implementation on the Internet of an active environment, upon which to rely for controlling the emergence of global behaviors.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced context-dependent coordination as a conceptual framework for the design and development of Internet applications based on mobile agents, and have shown how the associated concepts can be exploited in applications and can be implemented by relying on a proper coordination infrastructure based on programmable coordination media.

The availability of a framework in which agent coordination may easily be represented and controlled, and may be tuned to the specific needs of both applications and environments, positively impacts on application design and development, and may represent an important driving force towards the successful deployment of agent-based Internet applications. In fact:

- context-dependent coordination promotes a clear separation of concerns between intra-agent issues and inter-agent (coordination-related) ones. This makes application design more modular and less complex to be managed and, via a programmable coordination infrastructure, makes code less complex to write and easier to be maintained, even at run-time;
- by making coordination media programmable, administrators are enabled to customize their local coordination media to enforce local security and resource control policies, as well as to facilitate agents in accessing local resources and services;
- via a proper programming of the coordination media, agents can enforce application-specific coordination laws on the visited sites, such as the ones related to the execution of complex interaction protocols;
- a context-dependent coordination framework naturally matches the organizational perspective on software design promoted by agent-based computing.

Although some prototype infrastructures supporting context-dependent coordination – such as MARS – are already available, several issues not addressed by this paper still have to be analyzed. In particular:

(i) roles and role models [Ken00, Jen00], due to their importance in the analysis and design of multi-agent applications and, more generally, of organizations, have to be better integrated in the framework and in the corresponding coordination infrastructure;

(ii) more specific software-engineering methodologies must be defined, possibly by exploiting

assessed research results in the area of agent-oriented methodologies [AUML, MES, ZamJW01], to help in developing agent-based applications in the presence of mobility and context-dependency in coordination;

(iii) coordination infrastructures should integrate Agent Communication Languages technologies [Fin94, FIPA], to let agents interact in terms of high-level, knowledge-based, information. For instance, MARS could be extended to integrate tools automatically mapping ACL messages and protocols into tuples and reactions, respectively;

(iv) further emerging standards, other than Jini, must be taken into account. For instance, a coordination infrastructure for agents cannot disregard the current efforts in the area of XML-based data and service representation [GluTM99, CabLZ01], of CORBA-based interoperability [BelM01], as well as of agent interoperability [FIPA, Acities].

In addition, a more general and formal notion of context-dependent coordination, possibly defined by exploiting and extending already defined formal models for mobile systems [CarG00, PicMR00], is needed.

Acknowledgements: Work supported by the NOKIA Research Center Boston and by the Italian MURST within the project “MUSIQUE”.

REFERENCES

[AboM00] G. D. Abowd, E. D. Mynatt, “Charting Past, Present and Future Research in Ubiquitous Computing”, ACM Transactions on Computer-Human Interaction, 7(1):29-58, March 2000.

[Acities] The Agentcities Network, http://www.agentcities.org.

[AhuCG86] S. Ahuja, N. Carriero, D. Gelernter, “Linda and Friends”, IEEE Computer, 19(8):26-34, August 1986.

[AUML] The Agent Unified Modeling Language, http://www.auml.org.

[Bau98] J. Baumann, F. Hohl, K. Rothermel, M. Straßer, “Mole - Concepts of a Mobile Agent System”, The World Wide Web Journal,1(3):123-137, 1998.

[BelCS01] P. Bellavista, A. Corradi, C. Stefanelli, “Mobile Agent Middleware for Mobile Computing”, IEEE Computer, 34(3):73-81, March 2001.

[BelM01] P. Bellavista, T. Magedanz, “Middleware Technologies: CORBA and Mobile Agents”, in Coordination of Internet Agents, A. Omicini et al. (Eds.), Springer-Verlag, 2001.

[CabLZ00a] G. Cabri, L. Leonardi, F. Zambonelli, “Mobile-Agent Coordination Models for Internet Applications”, IEEE Computer, 33(2):82-89, Feb. 2000.

[CabLZ00b] G. Cabri, L. Leonardi, F. Zambonelli, “MARS: a Programmable Coordination Architecture for Mobile Agents”, IEEE Internet Computing, 4(4):26-35, July-Aug.

2000.

[CabLZ01] G. Cabri, L. Leonardi, F. Zambonelli, “XML Dataspaces for the Coordination of Internet Agents”, Journal of Applied Artificial Intelligence, 15(1):35-58, Jan. 2001.

[CarG00] L. Cardelli, A. D. Gordon, “Mobile Ambients”, Theoretical Computer Science, 240(1), July 2000.

[Cia98] P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, A. Knoche, “Coordinating Multi-Agents Applications on the WWW: a Reference Architecture”, IEEE Transactions on Software Engineering, 24(8):362-375, May 1998.

[DenNO98] E. Denti, A. Natali, A. Omicini, “On the Expressive Power of a Language for Programmable Coordination Media”, 10th ACM Symposium on Applied Computing, ACM Press, Atlanta, April 1998.

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

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

[FIPA] The Foundation for Intelligent Physical Agents, http://www.fipa.org.

[FreHA99] E. Freeman, S. Hupfer, K. Arnold, JavaSpaces: Principles, Patterns, and Practice, Addison-Wesley, 1999.

[FugPV98] A. Fuggetta, G. Picco, G. Vigna, “Understanding Code Mobility”, IEEE Transactions on Software Engineering, 24(5):352-361, May 1998.

[GluTM99] R. J. Glushko, J. M. Tenenbaum, B. Meltzer, “An XML-Framework for Agent-based E-commerce”, Communications of the ACM, 42(3):106-114, March 1999.

[IBM98] “T Spaces: the Next Wave”, IBM System Journal, 37(3):454-474, 1998.

[JaiAS99] A. K. Jain, M. Aparicio IV, M. P. Singh, “Agents for Process Coherence in Virtual Enterprises”, Communications of the ACM, 42(3):62-69, Mar. 1999.

[Jen00] N. R. Jennings, “On Agent-Based Software Engineering”, Artificial Intelligence, 117(2):277-296, 2000.

[KarT98] N. M. Karnik, A. R. Tripathi, “Design Issues in Mobile-Agent Programming Systems”, IEEE Concurrency, 6(3):52-61, July-September 1998.

[Ken00] E. A. Kendall, “Role Modeling for Agent Systems Analysis, Design and Implementation”, IEEE Concurrency, 8(2):34-41, April-June 2000.

[Klu01] M. Klusch, “Information Agent Technology for the Internet: A Survey”, Journal on Data and Knowledge Engineering, 36(3):337-372, March 2001.

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

[MES] The MESSAGE Methodology, Eurescom Project P907, http://www.eurescom.de/~public-webspace/p900-series/p907.

[MinU00] N.H. Minsky, V. Ungureanu, “Law-Governed Interaction: A Coordination & Control Mechanism for Heterogeneous Distributed Systems”, ACM Transactions of Software Engineering and Methodology, 9(3):273-305, July 2000.

[MosT95] Y. Moses, M. Tenneholtz, “Artificial Social Systems”, Computers and Artificial

Intelligence, 14(3):533-562, 1995.

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

[NorSR98] P. Noriega, C. Sierra, J. A. Rodriguez, "The Fishmarket Project. Reflections on Agent-mediated institutions for trustworthy E-Commerce", Workshop on Agent Mediated Electronic Commerce (AMEC-98), 1998.

[OmiZ99] A. Omicini, F. Zambonelli, "Coordination for Internet Application Development", Journal of Autonomous Agents and Multi-Agent Systems, 2(3):251-269, Sept. 1999.

[Par01] H. V. D. Parunak, S. Brueckner, J. Sauter, R. S. Matthews, "Entropy and Self-Organization in Multi-Agent Systems", 5th International Conference on Autonomous Agents, Montreal (CA), May 2001.

[Par97] H. V. D. Parunak, "Go to the Ant: Engineering Principles from Natural Agent Systems", Annals of Operations Research, 75:69-101, 1997.

[Pay02] T. R. Payne, M. Paolucci, R. Singh. K. Sycara, "Communicating Agents in Open Multi Agent Systems", 1st NASA Workshop on Radical Agent Concepts, Greenbelt (MD), Jan. 2002.

[PicMR00] G. P. Picco, A.M. Murphy, G.-C. Roman, "Software Engineering for Mobility: A Roadmap", in The Future of Software Engineering, A. Finkelstein (Ed.), ACM Press, pp. 241-258, 2000.

[PicMR99] G. P. Picco, A.M. Murphy, G.-C. Roman, "LIME: Linda Meets Mobility", 21st International Conference on Software Engineering, ACM Press, pp. 368-377, May 1999.

[SanH00] T. Sandholm and Q. Huai, "Nomad: Mobile Agent System for an Internet-Based Auction House", IEEE Internet Computing, 4(2):80-86, March-April 2000.

[Sat01] I. Satoh, "Flying Emulator: Building and Testing of Networked Applications for Mobile Computers", 5th International Conference on Mobile Agents, Atlanta (G), Dec. 2001.

[Smi80] R. G. Smith, "The Contract-Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", IEEE Transactions on Computers, 29(12):1104-1113, 1980.

[StaG90] J. W. Stamos, D. K. Gifford, "Implementing Remote Evaluation", IEEE Transactions on Software Engineering, 16(7):710-722, July 1990.

[SUN97] Java Servlet Development Kit (SUN), http://java.sun.com/products/servlet/index.html, 1997.

[Tol00] R. Tolksdorf, "Coordinating Work on the Web with Workspaces", 9th IEEE Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, Gaithersburg (MA), IEEE CS Press, June 2000.

[Wal97] J. Waldo, G. Wyant, A. Wollrath, S. Kendall, "A Note on Distributed Computing", Mobile Object Systems, Lecture Notes in Computer Science, No. 1222, pp. 49-64, Springer-Verlag, 1997.

[Wei93] M. Weiser, "Hot Topics: Ubiquitous Computing", IEEE Computer, 26(10), October 1993.

[Whi97] J. White, "Mobile Agents", in Software Agents, J. Bradshaw (Ed.), AAAI Press, pp.

437-472, 1997.

[ZamJW01] F. Zambonelli, N. R. Jennings, M. Wooldridge, “Organizational Rules as an Abstraction for the Analysis and Design of Multi-agent Systems”, International Journal of Software Engineering and Knowledge Engineering, 11(3):303-328, 2001.