Abstractions and Infrastructures for the Design and Development of Mobile Agent Organizations

Franco Zambonelli
Dipartimento di Scienze dell'Ingegneria – Università di Modena e Reggio Emilia
Via Vignolese, 905 – 41100 Modena – ITALY
Ph. +39-059-2056133 – Fax +39-059-2056126
franco.zambonelli@unimo.it

ABSTRACT

Internet applications can take advantage of a paradigm based on autonomous and mobile agents. However, suitable abstractions and infrastructures are required for the effective engineering of such applications. In this paper, we argue that a conceptual framework for context-dependent coordination, supported by an infrastructure based on programmable media, can promote a modular and easy to manage approach to the design and development of mobile agent applications in terms of computational organizations. The MARS coordination infrastructure is presented as an implementation of a coordination infrastructure promoting context-dependent coordination. A case study in the area of workflow management is introduced to clarify the concepts presented.

***Keywords**: Mobile Agents, Agent Organizations, Coordination Infrastructures, Agent-oriented Software Engineering.*

1 INTRODUCTION

Agent-based computing is being widely recognized as a useful paradigm for the design and development of complex software systems. Among the others, Internet applications such as E-commerce, information retrieval, distributed project-management, can effectively take advantage of the new paradigm.

As a rather young paradigm, agents and related features are still being analyzed and studied along several dimensions, one of which, *mobility*, is one of the most debated ones in the context of Internet applications. In most of the literature [Whi97, FugPV98], agent mobility refers only to the agents' capability of dynamically transferring their execution onto different Internet sites, with the goal of improving reliability and efficiency, and it is often highly criticized due to the technical problems it introduces. However, mobility is also a necessary abstraction to model the network-aware activities of Internet agents as well as the presence, in applications, of (software running on) mobile computing devices. In other words, mobility of agents is not only a useful feature, but it can be considered an intrinsic characteristic of Internet agents. As that, any methodology for the design and development of Internet-agent applications must exploit suitable abstractions, concepts, and infrastructures to deal with mobility.

One of the major problems introduced by mobility relates to the handling of the agents' coordination activities. First, when agents need to access data and services and to coordinate with other agents in a foreign Internet site, they are required to face all the typical problems of open systems and open organizations: heterogeneity of languages and protocols, and opportunistic behavior in interactions [Jen00, Zam01]. Second, each Internet site 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 (in other words, the environment can enact specific rules on the local organization and can bound agents to play a specific role in there [DemC96, FerG98]). Finally, agents that are part of a specific multi-agent application/organization may require their coordination activities to occur according to specific application needs, and/or to the specific role an agent is assigned in its original organization, despite the different characteristics of the Internet sites where they execute [Zam01].

In this paper, we focus on the conceptual framework of *context-dependent coordination*, firstly introduced in [CabLZ01], and show that it introduces suitable organizational abstractions to model the coordination activities of mobile application agents. The main idea is to abstract the "place" in which an agent execute and interact as a *local* and *active organizational context*. A local organizational context can actively enact specific rules on the agent's coordination activities, and different contexts may enact different, independent rules. Such a framework naturally invites thinking at mobile agents as organizational entities, moving across different organizational contexts, each enacting local organizational rules (or "social conventions") on local coordination activities. Moreover, by conceiving agents as capable of configuring (at least partially) the activities of interaction contexts, one can think at mobile agents as entities that belong to their original application/organization, and that can somehow enact application-specific organizational rules on their coordination activities, independently of their current position. Not only the organizational perspective of context-dependent coordination suits modern approach to agent-based computing, but it also promotes a clean separation of concern in the analysis and design of mobile agent application. In fact, the analysis and design of application can clearly separate the intra-agent issues, related to the internal computation activities of application agents, from the inter-agent ones, related to the management and control of the agents' coordination activities in an organization.

Of course, as is it the case of any model, the potentials of context-dependent coordination can be fully exploited in application development and execution only via the availability of proper infrastructures. In particular, context-dependent coordination can be effectively supported by a coordination infrastructure based on a multiplicity of independent programmable coordination media [DenNO98, CabLZ00], each associated to an Internet site. By implementing the concept of active interaction contexts, programmable coordination media become the places where to embed the code implementing organizational rules, whether statically imposed by site administrators to rule activities in local organizations, or dynamically programmed by mobile application agents when arriving at a site. Such an infrastructure enables an easy translation of a context-dependent design into actual code, and promotes the writing of modular, easy to change and maintain, code.

In the following of this paper, we go into details about agent mobility, context-dependent coordination, and their impact on application analysis and design (Section 2). Then, we analyze the main issues underlying programmable coordination infrastructures, describing an actual implementation of such an infrastructure, i.e., MARS, and analyzing their impact on application development (Section 3). We introduce a simple case study to clarify the concepts expressed (Section 4). Finally, we discuss related work in the area (Section 5) and conclude the paper by outlining promising research directions (Section 6).

2 CONTEXT-DEPENDENT COORDINATION

2.1 Types of Mobility

Mobility of agents may appear in different flavors in today's Internet computing environments:

- *Virtual agent mobility*. The wideness and the openness of the Internet scenario makes it suitable to develop applications in terms of *network-aware* agents, i.e., agents that are aware of the distributed nature of the target and explicitly locate and access resources, services, and agents. From a different perspective, agents "navigate" the Internet and virtually move across its resources [Wal97, CarG00].
- *Actual agent mobility*. This refer to the agents' capability of moving across Internet sites while executing by dynamically and autonomously transferring their code, data, and state, toward the resources they need to access [FugPV98].
- *Physical agent mobility*. Mobile devices accessing the Internet – such as palmtop, cellular phones, etc. – can be modeled in terms of autonomous agents whose position with respect the fixed network infrastructure (and thus the point from which to access it) changes accordingly to their geographical movements (that, from the point of view of the fixed infrastructure, are perceived as autonomous movements).

Virtual agent mobility is a useful abstraction to be exploited in the design of Internet-agent applications. Although it is possible to think at network-aware agents as autonomous components that have the possibility of globally interacting with a world-wide environment from a fixed perspective, minimization of conceptual complexity suggests conceiving network-aware agents as localized, situated, entities, that moves across a world-wide network of independent sites, no matter if their execution is tied to a fixed Internet node [Wal97].

Actual agent mobility is a useful dimension of autonomy for the effective execution of application agents. In fact, despite the technological challenges (in terms of security and resource control) that agent mobility introduces, its wise exploitation can provide for saving network bandwidth, increasing both the reliability and the efficiency of the execution, handling mobility of users [Whi97, LanO98]. Moreover, even if similar advantages and functionality of actual agent mobility can be exploited by weaker forms of low-level mobility (e.g., code mobility [FugPV98]), only agent mobility can naturally map at the execution level the virtual mobility already exploited at the design level.

Physical mobility is likely to become an increasing issue to be taken into account in the design of Internet applications, as more and more there will be the need of mobility-enabled services and applications. Many applications and services that we currently design and develop for being exploited via non-mobile Internet-connected devices are likely to be exploited in the near future in a mobile setting. Devices will become mobile and will be in need of accessing the Internet from different – and dynamically changing – access points. Thus, all software agents that are currently non-mobile will become mobile agents simply for the fact of executing in a mobile device. Moreover, even if no agent-technology is exploited inside a mobile device, the observable behavior of a mobile-device w.r.t. the fixed infrastructure is actually those of an autonomous (in terms of mobility and proactivity) component, i.e., of a mobile agents, and will have to be modeled accordingly.

2.2 Local Interaction Contexts

To deal with all the above types of mobility in a uniform way, a suitable modeling of the scenario in which applications execute is required. In particular, it is suitable to model and develop Internet applications in terms of agents that can autonomously move across a multiplicity of different sites or environments, e.g., Internet nodes or administrative domains of nodes (see Figure 1). A site defines the *context* in which an agent executes and interacts. In other words, a site acts both as *execution context*, i.e., the place in which agents executes, whether virtually, actually, or physically, and as *interaction context*, i.e., the place in which agent coordination activities occur. Coordination activities may include accessing the local resources of a site and communicating and synchronizing with other locally executing agents, whether belonging to the same application or foreign Internet agents.

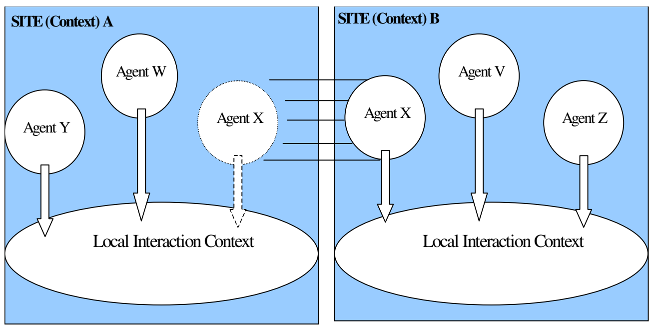

Figure 1. Mobility across interaction context

What the interaction model to be actually exploited for modeling interaction is not of primary influence. Interaction in a context may occur via message passing (and, at a higher-level, via ACLs [Fin94]), via event-channels [Bau98], via meetings [Whi97], or via shared information spaces [CabLZ00]. The choice of a specific interaction model can be determined both by the specific application needs and by the available communication and coordination infrastructure. What really matters from the software engineering perspective is the locality model of interaction enforced by local interaction spaces, which naturally reflects – at the level of application modeling – a notion of context which is intrinsic in mobility. In fact, for the very fact of moving across the Internet, agents will access different data, services, and will interact with different agents, depending on their current position, i.e., of their current interaction context.

2.3 From Interaction Contexts to Organizational Contexts

The above form of *context-dependency*, intrinsic in agent mobility and made explicit by a modeling of the Internet as a multiplicity of interaction contexts, does not imply interaction contexts to play any active role in it. In fact, an interaction context may act as a simple data and service repository, or as an event and message forwarder. In other words, an interaction context is simply a *place* rather than an *organization*. However, more sophisticated forms of context-dependency may be needed in Internet applications suggesting an active role of interaction contexts and leading to an *organizational perspective*:

- each context has its own specific characteristics and security policies, and it may be somehow in need of enacting specific coordination laws to rule the coordination activities of the locally executing agents (*environment-dependent coordination*). These coordination laws may include security and resource control policies, as well as policy to rule inter-agent interactions and control the evolution of inter-agent coordination protocols.

- application agents belonging to a specific multi-agent application may require their coordination activities to occur according to specific application needs, e.g., agent may wish to each specific coordination laws on their coordination activities, independently of the interaction context in which they actually take place and on the coordination laws there enacted (*application-dependent coordination*).

With regard to the first point, as all coordination activities of mobile agents on a site occur via the local interaction context, it is conceptually wrong to think at local interaction policies and coordination laws as something unrelated to interaction contexts. Instead, by assuming an organizational (or social) perspective, one can consider the interaction context in terms of an organizational (or social) context. In other words, one must consider that an agent, by entering via migration an interaction context, it enters a foreign organization (society) in which specific organizational rules (or social conventions) are likely to be intrinsically enacted in the form of coordination laws. Thus, for the sake of conceptual simplicity, one must consider the interaction context as the locus in which the organizational rules ruling the activities of the local organization reside [ZamJW01].

With regard to the second point, one must take into account that mobile agents executing within a distributed multi-agent applications, still logically belong to the organization (society) defined by the multi-agent application in itself, despite the fact that they may are currently executing in a foreign organization. In other words, the context in which an agent executes and interacts is not only the one identified by the local interaction context, but is also the one of its own organization. As that, a local interaction context should not only be though as the place in which local organizational rules reside, but also as a place in which application agents may enact their own, application-specific, organizational rules in the form of coordination laws.

Keeping all together, the introduced concepts lead to the scenario depicted in Figure 2, which represents a usable a modular conceptual framework for the analysis and design of mobile agent applications. On the one hand, interaction contexts associated to different sites may enact specific organizational rules and, thus, may exhibit different behaviors in response to the same interaction events. In an organization/site, agents are forced to play a specific role and their coordination activities occur accordingly to local, site-specific organizational rules. On the other hand, since agents also execute in the context of their original application/organization, coordination activities will be influenced not only by the local organizational rules, different from site to site. Instead, application-specific organizational rules can be enacted for all the agent of that application on any visited site, to act concurrently with the local organizational rules.

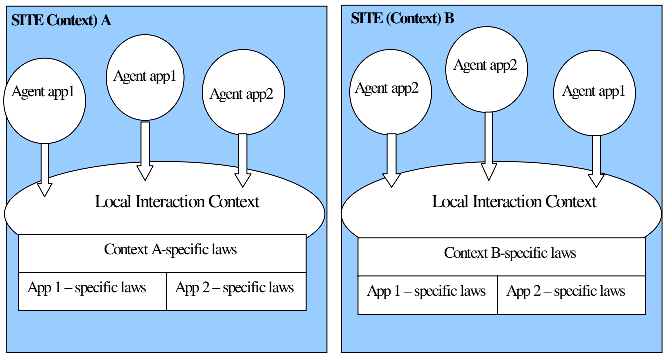

Figure 2. Context-dependent coordination: The framework

2.4 Impact on Analysis and Design

From the point of view of application analysis and design, context-dependent coordination represents a useful conceptual framework. In fact, it naturally invites in analyzing and designing an application by clearly separating the *intra-agent* issues – related to the specific computational roles of the application agents – and the *inter-agent* ones – related to the interaction of the agents with the other agents of the same application and with the visited execution environments. In other words, context-dependent coordination promotes a clear separation of concerns, which is likely to reduce the complexity of application analysis and design.

Without the ambition of introducing a detailed methodology, we can sketch the general guidelines for the identification of intra-agent and inter-agent issues. These guidelines represent the adaptation to the area of mobile Internet agents of a previous study in the more general area of agent-oriented software engineering and methodologies [Zam01, ZamJW01].

From the point of view of application developers, the analysis phase can be organized as follow:

- at the intra-agent level, one has to analyze the global application goal and decompose it into sub-goals. This process should lead to the identification of the *roles* to be played in the application [WolJK00];

- at the inter-agent level, one should identify of how roles relate to each other, and of how interaction between roles must occur for the global application goal to be correctly achieved. This process should lead to the identification of the organizational rules, driving the whole behavior of the agent organization [ZamJW01].

The following design phase should build on the outcome of the analysis phase. One the one hand, accordingly with the guidelines of the Gaia methodology [WolJK00], the design phase should identify the agent classes that will form the application, and the agent instances that have to be instantiated from these classes. The agent of a class can play more roles, and more agents of a class can be instantiated to play the same roles. On the other hand, the identified application-specific organizational rules must be used to precisely define, in operative terms, the way coordination activities should be ruled [ZamJW01]. This amount at defining the protocols to be used by agents to interact with each other as well as those to be used to access to the resources of the visited sites. In addition, this imply defining the constraints under which the protocols can be executed and the computational activity that must be issued in interaction contexts to properly mediate and support the execution of protocols.

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 infrastructure) the administrator of one site can identify all the local organizational rules that he may find it necessary to enact. These rules can be used to facilitate the execution of the agents on the site, and/or to make the structure of the local organization homogeneous to agents' expectations (or to the requirements of the forming federation), and/or to protect the site from improper exploitation of the local interaction context. Again, the local organizational rules identified by the analysis phase are to be used in design to define protocols in operative terms and to define the associated computational activity to be performed in the local interaction context.

3 PROGRAMMABLE COORDINATION INFRASTRUCTURES

Context-dependent coordination not only introduces a suitable, organization-oriented, framework for the analysis and design of mobile agent applications, but it also represent a reference model for an Internet coordination infrastructure suited to the needs of mobile agent applications.

3.1 Programmable Coordination Media

For a coordination infrastructure to enforce context-dependent coordination, it must somehow implement the abstraction of local programmable interaction contexts via an architecture based on local and *programmable coordination media*.

In general terms, a coordination medium is the software system enabling interaction and coordination, accordingly to a given interaction model, among a set of agent/processes executing within a locality. Whatever the interaction model defined, a coordination medium is characterized by: *(i)* a set of operations to be used by agents to access it and *(ii)* an internal behavior, intended as the computational activities performed inside the coordination medium in response to access events. For instance, with reference to the tuple space model, a coordination medium is characterized by *(i)* a limited set of primitive operations to store, read and consume tuples from the tuple space and *(ii)* an internal behavior based on a blocking pattern-matching access to tuples. In particular, an agent can request a tuple by providing a template; the tuple space returns to the agent one tuple matching the template, if available, blocks the requesting agent otherwise.

Most of the communication and coordination infrastructures (as well as most of available tuple space implementations) fix the internal behavior of coordination media once and for all. *Programmable coordination media* [DenNO98, MinU00], instead, make it possible – without changing the set of primitive operations used by agents to access coordination media – to program the internal behavior of the coordination media in response to access events accordingly to the specific needs of the applications or of the local environments. For instance, let us 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 (class of) access events – i.e., requests for storing, reading, or extracting given tuple classes performed by a given classes of agents – a reaction different from the basic pattern-matching mechanism to be programmed and issued in response to it.

3.2 The MARS Infrastructure

MARS (*M*obile *A*gent *R*eactive *S*paces), developed at the University of Modena and Reggio Emilia and described in detail in [CabLZ00], implements a programmable tuple-based coordination infrastructure for Java mobile agents. The characteristics of MARS and of its programmable tuple space model make it a suitable infrastructure for context-dependent coordination.

Globally, the MARS architecture is made up of a multiplicity of independent programmable tuple spaces. One tuple space is typically associated to a node and can be accessed only by the agents locally executing in that node (see Figure 3). Thus, MARS is explicitly conceived for actually mobile agents, in that to access a MARS tuple space agents have to actually execute on the corresponding node/domain. When an agent arrives on a node (or on a domain of nodes sharing a single MARS tuple space), it is automatically bound to the local MARS tuple space. To strictly enforce a context-dependent coordination model, the MARS tuple space on a node is intended as the only resource an agent can access on that node, and it should be therefore exploited for both inter-agent and agent-environment coordination.

The extension of MARS to support virtually mobile agents and physically mobile ones is under development.

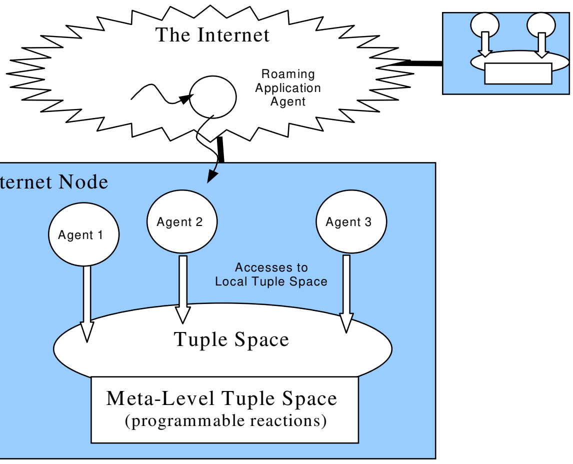

Figure 3. The MARS Architecture

The MARS interface to access a tuple space has been designed in compliance with the Sun's *JavaSpaces* one [FreHA99]. Tuples and template tuples (both usually called "Entries") are Java objects whose instance variables, possibly non-defined 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 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 model. 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*.

The programmable model of MARS enables the association of any needed reactions in response to access events performed by agents. This association occurs via *4-ples* 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 (representing 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*. Putting and extracting tuples from the meta-level tuple space corresponds to installing and uninstalling, respectively, reactions associated to events at the base-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.

A meta-level 4-ple can have some non-defined values, in which case it associates the specified reaction to all the access events that match it. For example, the 4-ple (Reaction, null, read, null) in the meta-level tuple space associates the reaction of an object instantiated by the Reaction class to all read operations, whatever the tuple type and content and the identity of the agent performing the operation. When an access event to the base-level tuple space occurs, MARS issues the pattern-matching mechanism in the meta-level tuple space to look for reactions to be executed in response to the access event. If several 4-ples satisfy the meta-level matching mechanism, all the corresponding reactions are executed in a pipeline, accordingly to their installation order (or to a specific order determined by the administrator). When a reaction method executes, it is provided with the identity of the agents that has triggered the reaction, the operation it has performed, the tuple provided as parameter in the operation and the tuples returned in the reaction previously executed in the pipeline, if any.

The fact that a reaction can be associated to access events either when performed by agents with a specific identity or independently of the identity of the accessing agents enables for the embedding in the form of reactions in the tuple space of both local coordination laws and of application-specific coordination laws. 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 nodes, and these reactions have to apply only to those agents belonging to the same application context. Since an agent must not be allowed to influence the coordination activities occurring in a different application contexts, specific security mechanism in MARS guarantee that agents, unless explicitly authorized, can influence only their local application context.

3.3 Impact on Development

Given the availability of a programmable coordination infrastructure, the separation of concerns between intra-agent issues and inter-agent, organizational, ones, enforced during analysis and design, can be preserved during the development and maintenance too.

If, as in MARS, the code of the agents can be clearly separated from the code implementing the coordination laws (representing local and/or application-specific organizational rules), 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.

The advantages of context-dependent coordination and of programmable coordination infrastructures make us strongly encourage their adoption. However, we also want to emphasize that, in the presence of an infrastructure based on programmable coordination media, neither the designer, nor the developers, nor the administrators of one site are obliged to commit to a context-dependent approach. They are fully free to neglect the programmability of the coordination media, develop applications in terms of agents interacting via non-programmed coordination media, and exploit applications already developed for such infrastructures.

4 A CASE STUDY

As a simple case study in the area of workflow management, we consider the review process of a big international conference. The engineering problems related to the agent-mediated management of such a process has been analyzed in detail in [Zam01, ZamJW01]. Here, after having presented the case study in general terms, we focus on a few issues related to the management of the coordination activities of mobile agents.

The review process of a big international conference can be based on a partitioning of the submitted papers among PC members (or among the members of a steering committee). Each PC member is in charge of appropriately recruiting referees for its paper partition, of collecting the required number of reviews for each paper, and of sending them back to the PC chair. Referees, by their side, are in charge of retrieving papers to review and of filling the associated review form by the stated deadline.

Let us now consider the distributed scenario of a mobile-agent-mediated management of review process activities, by assuming the presence of the MARS infrastructure. The PC chair is supposed to make available all submitted papers in the form of tuples in its local MARS tuple space. If the PC Chair gives to PC members the freedom of choosing which papers they should take care of, a PC member can exploit mobile agents to go to the PC Chair site and there select the required number of papers on the basis of personal preference criteria. PC member agents can copy these papers in the local PC member's MARS tuple space, which will also acts as a repository for both white and filled review forms, and will decide about the assignments of papers to referees. A referee, at its turn, will send a mobile agent to the site of the PC member that has recruited him, to retrieve the papers assigned to him (or to negotiate these assignments) and the associated review forms. Once the review is completed, the mobile agent of the referee is sent back to the PC member site to put in it the filled review forms, where PC member agents can eventually analyze the outcome of the review process.

4.1 Analysis

By relying on the conceptual framework of context-dependent coordination, and keeping into account the application developers' perspective, the analysis of the above case study introduces the issues of identifying the different roles to be played in the application and the organizational rules. In particular, after the analysis of the case study, one could have identified the different roles of "collector of papers for a PC member" and "referees recruiter", and the single role of "referee". Moreover, the analysis should have also identified organizational rules that must be respected in the review process activities. For instance, two of such rules may specify that *(r1)* the referee recruiter cannot assign more than *N* papers to the same referee, and that *(r2)* a referee cannot read a paper unless it is assigned to them. Further rules can be defined along the guidelines that traditionally rule the review process of a conference.

From the point of view of system administrators, the analysis should identify the possible "attacks" to the local system that can be performed by malicious application, and the problems that agents can encounter in accessing local resources. For example, the administrator of a PC member site can be required to store all local technical reports and documents, there included the submitted paper, in the "pdf" format. If papers have to be submitted to the conference in "ps" format, and application agents will accordingly look for "ps" files, a local organizational rule could then be the one that specify that *(r3)* requests for "ps" files must be considered as valid requests for "pdf" files.

4.2 Design

The outcome of the analysis phase should be used to drive the design of the application, aimed at identifying which agent classes should be defined and which agents should be instantiated from them. In the case study, a reasonable choice would be to collect in a single PC member agent class all the roles actually belonging to PC members, i.e., "collector of papers" and "referees recruiter". Instead, the role of "referee" would reasonably be implemented as a single agent class.

In addition to the identification of agents, the design phase in our approach also requires defining which behavior should be actually programmed by application agents in the interaction context so that all application protocols can execute in respect of the identified organizational rules. With regard to the organizational rule *(r1)*, it implies associating a behavior to interaction context, which avoids a PC member to involve a referee in a protocol/negotiation for the assignment of a paper more than *N* times. In a similar way, and from the point of view of system administrators, rule *(r3)* implies associating a behavior that is able to transform any protocol for the request of a "ps" file into a protocol for the request of a "pdf" file.

4.3 Development

The development phase simply reduce at separately coding agent classes and organizational rules (in the form of coordination laws). With reference to MARS and Java-based mobile agents, we sketch some code fragments for agents and coordination laws.

Figure 4 shows a fragment of the code of PC member and referee agents, and in particular the code associated with the protocol for assigning papers to referees. The PC member agent analyzes its paper, chooses a referee, and asks him for its willingness to review the paper. The referee agents goes to the site of the PC member, checks which papers the PC member has assigned to it, and decide if it want to review them or not. All this protocol occurs via exchanging of tuple in the MARS tuple space of the PC member. Although the figure shown agents implemented as simple Java thread, nothing in our approach and in the MARS infrastructure prevents from adopting any other suitable agent architecture. The only requirement is that agents interact with accordingly to the MARS interface.

Figure 5 shows the code of a Java class, in the format of MARS reactions, implementing the organizational rule *(r1)*. This simply amounts at avoiding the PC member writing more than N AssignmentTuples with the same referee ID as field. By writing the meta-level tuple:

(LimitNumber,PC-member-ID, write, AssignmentTuple(null,null))

in the tuple space, the reaction will be triggered whenever a PC member agent writes an AssignementTuple in the tuple space. Figure 6 shows the code implementing the organizational rule *(r2)*, to avoid a referee agent to read a paper unless that paper has been explicitly assigned to it. Figure 7 shows the code implementing the organizational rule *(r3)*, which transforms any request for a (tuple representing a) "ps" file info a request for a (tuple representing a) "pdf" file.

4.4 Discussion

Not only, as already stated, the code for agents and organizational rules can be changed, and re-used independently of each other, but also new organizational rules can be added and old can be dismissed by influencing neither the behavior of agents nor the one of organizational rules.

```
// Excerpt of the class for the PC Member agents
class PCMemberAgent extends Agent {
private Space Local;
private PaperTuple PaperPattern, Paper; Identity ChosenRef;

public void run()
{ …
// creation of template tuple - class PaperTuple representing papers
PaperPattern = new PaperTuple(null, null, null, null, null, "ps");
// retrieve from the space (via a read operation) a paper
Paper = Local.read(PaperPattern, null, NO_WAIT);
//analyse the paper on the basis of its characteristics
// and decide about a proper referee
// put in the space a tuple of class AssignmentTuple
// to specify the association paperID-refereeID
Local.write(new AssignTuple(Paper.Paper,ID,ChosenRef),null, 0);
// wait for a tuple of class AckTuple from the referee
Local.take(new AckTuple(Paper.PaperID,ChosenRef,null),…);
… // go on with the process
}

// Excerpt for the class for the referee agents
class RefereeAgent extends Agent {
private Space Local; …

public void run()
{…
// the agent migrates to the PC member site
this.go_to("PCChair.site.com");
// note: that could also be a "virtual" migration
// creation of a template tuple – class AssignmentTuple
AssTupPattern = new AssignementTuple(null, MyID);
// retrieve from the space all AssignementTuples
AllAssigned = Local.readAll(AssTupPattern, null, NO_WAIT);
// checks, for each assigned paper, which it can referee,
// if the i-th paper assigned can be refereed it is read from the space
Local.read(new PaperTuple(AllAssigned[i].PaperID, null, null,..)…);
// the corresponding ack tuple is put in the space
Local.write(new AckTuple(AllAssigned[i].PaperID, MyID, YES);
…// go on with the process
}
```

Figure 4. A Fragment of the code of the PC Member and the Referee agents

With reference to the protocol via which a referee access to papers, one can see that code of figure 6 naturally pipelines with the one of figure 7, leading to a composition of the local and application-specific organizational rules associated to the execution of that protocol. Once the local organizational rule select "pdf" papers on the basis of the "ps" request performed by agents, the application-specific rule can select, among all papers, only those that have been actually assigned to the requesting agents. Let us now suppose that the conference official decide that even those referee that are not in charge of the review of a paper can take a look at it. Then, the application can be simply updated by de-installing the code implementing the organizational rule *r2* (i.e., with reference to MARS, by simply deleting the associated meta-level tuple).

In a similar way, new organizational rules can be added and further pipelined with existing ones. For instance, let us suppose that the conference officials decide that a single referee cannot review more than one paper of the same author. This influences the protocol for assigning papers to referees, and should avoid a PC member to involve a referee in the paper assignment protocol if that referee has been previously involved in the same protocol for a paper of the same author. This new organizational rule can be coded by a MARS reaction that reacts to all insertions of AssignmentTuples performed by PC members, and checks them against the new organizational rule. Such a reaction can then be installed in the MARS meta-level tuple space to take effect, and it can be pipelined with the reaction of Figure 5, implementing rule *r1*.

```
class LimitPapers implements Reactivity
{
  public Entry[] reaction(Space s, Identity Id, Operation Op, Entry
Template, Entry InputTuples[])
  { Identity CurrAssTo; AssignmentTuple PastAss[];
  CurrAssTo = (AssignmentTuple)Template.CurrentRef;
  // retrieve all tuple representing previous assignments
   // to the same referee
  PastAss = s.readAll(new AssignmentTuple(null,CurrAssTo, ..)..);
  // if there are less than N, also the current assignemment
  // can be written as a tuple
  if(PastAss.lenght()<N) s.write(Template, null, 0);
  return null;
} }
```

Figure 5. Reaction limiting the paper assignments to a referee

```
class PS2PDF implements Reactivity
{
  public Entry[ ] reaction(Space s, Identity Id, Operation Op, Entry
Template, Entry InputTuples [ ])
  // no match if the site stores the extension as "ps"
  { // thus modifies the extension of the required files
    ((PaperTuple)Template).Format = "pdf";
    // match with pdf extension
    Entry [ ] result = s.readAll(Template, null, NO_WAIT);
    return result;
  } }
```

Figure 6. Reaction transforming request for "ps" files into requests for "pdf" files.

```
class ReadOnlyAssigned implements Reactivity
{
  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++)
    // for each paper, checks if its is assigned to the agent
    // i.e., if a corresponding AssignmentTuples exists
    if (!s.read(new AssignmentTuple(InputTuple[i].PaperID, Identity)))
            // the paper tuple has not to be returned
            InputTuple[i]=null;
      // return only the updated tuples (null elements are purged)
    return compactArray(InputTuples);
} }
```

Figure 7. Reaction enabling a referee to read only those papers assigned to it

From the agent point of view, provided that the protocol interfaces remain unchanged from the agent point of view, one can change the code of the agents without necessarily changing the organizational rules. In the specific example, one can change the code that drive PC member agent toward the choice of a specific paper-referee assignment without influencing at all other the code of the coordination laws.

Clearly, the above considerations about re-use and maintenance of code also apply at the analysis and design level. In fact, due to the separation of concern enforced in the whole software development process, the analysis and design phases benefits from the possibility of separately modifying and updating the outcomes of inter-agent and intra-agent analysis/design.

5 RELATED WORK

A few of the proposals in the area of Internet agents explicitly focus on models and coordination infrastructures suited to mobility. However, models and systems, in different areas, focus on problems (and propose solutions) strictly related to the ones discussed in this paper.

5.1 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, ZamJW00], "social laws" [MosT95], "social conventions" [NorSR98], "active environments" [Par00] are receiving more and more attention, and are leading to a variety of models and proposals. 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 organization/society in which agents live and have to interact, and the associated organizational/societal laws.

As an example, several researches show that engineering a complex multi-agent organization does not simply require the identification of the roles to be played by agents, but it 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 [ZamJW00]. As another example, in ant-based systems complex global behaviors can emerge from the interactions of very simple agents with the environment [Par00]. 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 depends strictly 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.

The framework of context-dependent coordination is likely to facilitate the definition and the implementation of complex multi-agent systems. In fact, as we have shown, organizational rules can be easily mapped in terms of application-specific coordination laws and can be enforced by a proper programming of coordination media, without having to hardwire them into agents or into the organizational structure. Moreover, a programmable coordination medium represents the most natural implementation of an active environment upon which to rely for controlling the emergence of global behaviors.

5.2 Coordination Infrastructures

The *T Spaces* project at IBM [IBM98] and the *JavaSpaces* project at Sun [FreHA99] define an infrastructure based on 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, although T Spaces integrates a limited form of programmability, by enabling complex queries to be added as primitives to a tuple space.

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

The LIME infrastructure faces the problem of handling interactions in the presence of any kind of active mobile entities [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. 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. 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 system for agent-mediated auctions [NorSR98], with the goal of guaranteeing that auctions can proceed accordingly to specific auctions' conventions. Although not explicitly addressing agent mobility issues and application-dependent coordination issues, both the above approaches are suitable to enforce local coordination laws in systems of peer-to-peer interacting agents.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced context-dependent coordination as a conceptual framework for Internet applications based on mobile agents and have shown how it can promote a modular – organization-oriented – approach to analysis and design. Moreover, we have shown how a programmable coordination infrastructure, by mapping at the infrastructure level the abstractions of context-dependent coordination, can lead to the development of easy to program and easy to maintain mobile agent organizations.

In the future, we intend to develop a formal model of context-dependent coordination, possibly by exploiting already defined models for mobility and mobile agents [CarG00, PicMR00]. In addition, we indent to better investigate and integrate in our model role modeling techniques [Ken00, WolJK00] and agent communication languages [Fin94].

REFERENCES

[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.

[CabLZ00] 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, "Engineering Mobile Agent Applications via Context-Dependent Coordination", 23rd International Conference on Software Engineering, IEEE CS Press, May 2001.

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

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

[DenNO98] E. Denti, A. Natali, A. Omicini, "On the Expressive Power of a Language for Programmable Coordination Media", Proceedings of the 10th ACM Symposium on Applied Computing, ACM, 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.

[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.

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

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

[JenW00] N. R. Jennings, M. Wooldridge, "Agent-Oriented Software Engineering", in *Handbook of Agent Technology*, J. Bradshaw Ed., AAAI/MIT Press, 2000.

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

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

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

[MosT95] Y. Moses, M. Tenneholtz, “Artificial Social Systems”, *Computers and Artificial Intelligence*, 14(3):533-562, 1995.

[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.

[Par00] H. V. D. Parunak, S. Brueckner, J. Sauter, R. S. Matthews, “Distinguishing Environmental and Agent Dynamics ”, 1st International Workshop on Engineering Societies in the Agents' World, Lecture Notes in Computer Science, No. 1972, Springer-Verlag, 2000.

[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, pp. 368-377, May 1999.

[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, Berlin (D), 1997.

[Whi97] J. White, “Mobile Agents”, in *Software Agents*, J. Bradshaw (Ed.), AAAI Press, pp. 437-472, 1997.

[WolJK00] M. Wooldridge, N. R. Jennings, D. Kinny, “The Gaia Methodology for Agent-Oriented Analysis and Design”, *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285-312, 2000.

[Zam01] F. Zambonelli, N. R. Jennings, A. Omicini, M. J. Wooldridge, “Agent-Oriented Software Engineering for Internet Applications”, in *Coordination of Internet Agents*, A. Omicini et al. (Eds.), Springer-Verlag, 2001.

[ZamJW00] F. Zambonelli, N. R. Jennings, M. J. Wooldridge, “Organizational Abstractions for the Analysis and Design of Multi-agent Systems”, in *Agent-Oriented Software Engineering*, LNCS No. 1947, November 2000.