Developing Mobile Agent-Based Applications

Fabiana P. Guedes, Patrícia D. L. Machado and Vivianne N. Medeiros[1]
Federal University of Campina Grande, Systems and Computing Department
Campina Grande, Brazil, 58109-970
{fabianap,patricia,vivianne}@dsc.ufcg.edu.br

Abstract

Issues related to developing distributed applications based on mobile agents are discussed. Up to now, the majority of existing mobile agent-based applications have been created in an *ad-hoc* way, following little or no methodology. One reason is that current development models do not properly cover requirements and aspects of mobility in the modeling, designing and verification of such applications. We present a methodology of developing mobile agent-based applications that is based on the standard iterative and incremental unified process, the use of mobile agent design patterns and issues that should be considered in the activities of analysis and design. Artifacts are produced using an extension of the Unified Modeling Language (UML) that copes with mobility. A case study is presented to illustrate the applicability of the model.

Keywords: mobile agents, unified process, UML, design patterns, distributed applications.

[1] This work and its authors are supported by CNPq, process 552190/2002-0. First author was supported by CAPES.

1 Introduction

Agents are software units that can execute particular tasks and deal with environmental changes through autonomy, intelligence, mobility, cooperation and reactivity. Particularly, mobile agents are autonomous software entities that can migrate to different physical locations and continue its execution at the point where it stopped before migration. Regarding distributed applications, the main advantages promised by mobile agents are reduction of network traffic, load balancing, fault tolerance, asynchronous interaction, data access locality and flexible distribution of intelligence in a network [13,18,12]. The growing interest in mobile agent technology has been motivated by its potential use in a number of application areas including electronic commerce, information gathering and dissemination, telecommunication systems, customizable services and monitoring systems [18,9,23].

Mobility provides clear advantages as a programming concept as well as a software development technology. Several mobile agent platforms have been developed, particularly based on the Java Language [13,18,10]. Also, a number of case studies have demonstrated the applicability of mobile agents [9,23,1,12,6]. Nevertheless, up to now, the majority of existing mobile agent-based applications have been created in an *ad-hoc* way, following little or no methodology. One reason is that current development models do not properly cover requirements and aspects of mobility in the modeling, designing and verification of such applications. Clearly, software agents impose profound chances in current software development methodologies and techniques [24,19,12].

Some approaches toward a methodology for developing multi-agent systems focusing on patterns of interaction and cooperation have already been proposed [24,20,19]. Nevertheless, mobility has not yet been appropriately considered in a software development process as a whole, even though some important attempts focusing on specific activities, techniques and notations have already been made. The use of components for modeling mobile agents is explored in [25]. An abstract notation for modeling mobile applications is presented in [3]. The approach presented in [12] focus on modeling mobile agent systems using an extension of UML. The concept of context dependent coordination for mobile agents to simplify design and help deployment of Internet applications is introduced in [2]. Tahara et al [22] proposes that design be split into two activities: platform independent architectural design and platform dependent detailed design, where design patterns are applied at appropriate abstract levels. Nevertheless, more complete approaches covering thoroughly disciplines of analysis, design, implementation and testing are still needed as well as substantial case studies using such approaches.

In order to contribute for a practical use of mobile agent concepts and technology according to a software engineering discipline, the main objective of this work is to provide a methodology of developing mobile-agent based applications. The methodology is intended for being used as part of process models based on the standard iterative and incremental unified process [11,14]. Also, the methodology applies mobile agent design patterns as suggested by Tahara et al [22] and an extension of UML that copes with mobility given by Klein at al [12]. Moreover, the methodology addresses issues that should be considered in the activities of analysis and design. Since mobile agent-based applications are usually built on different platforms and current platforms are evolving rapidly, detailed design is also split into two activities: platform independent and platform dependent. The former focus on the detailed logical design of the application whereas the second focus on detailed design decisions related to a specific platform chosen to implement the application. A case study is used to illustrate the main ideas presented.

The paper is structured as follows. Section 2 describes the methodology proposed and its main phases and disciplines. Platform dependent design is based on the Grasshopper platform [10]. Section 3 presents a case study focusing on the activities of analysis and design. Finally, Section 4 presents concluding remarks and pointers for further work. We assume the reader to be familiar with basic concepts of mobile agents as presented in [7] and the unified process [11].

2 A Development Methodology

In this section, we introduce a methodology of developing mobile agent-based applications that is based on the widely used unified process, focusing on the use of UML and design patterns. Moreover, it applies and extends the ideas presented by Tahara et al [22] on the design of mobile agent-based applications by applying patterns according to specific architectural levels, where, to maximize reuse, higher levels are independent of specific agent platforms. An extension of UML proposed by Klein et al [12] is used to allow for a proper abstract modeling of aspects of mobility. Figure 1, presents the model to be followed and its main disciplines. The model is iterative and incremental, i.e., it is composed of a number of iterations over the time that produces an incremented version of the system.

The need for a model of developing mobile agent-based applications is evident. It is been widely accepted that current methodologies and techniques are not suitable for developing agent-based applications [24,20,19,5]. Modeling and designing such applications involve additional issues and decisions not faced in ordinary object-oriented software development and even conventional client-server systems development. At design level, specific

mobile agent patterns can be used to represent solutions to be implemented in different agent platforms. Additional concepts have to be dealt with: mobile agent, stationary agent, agency and region, agent mobility, cloning and so on.

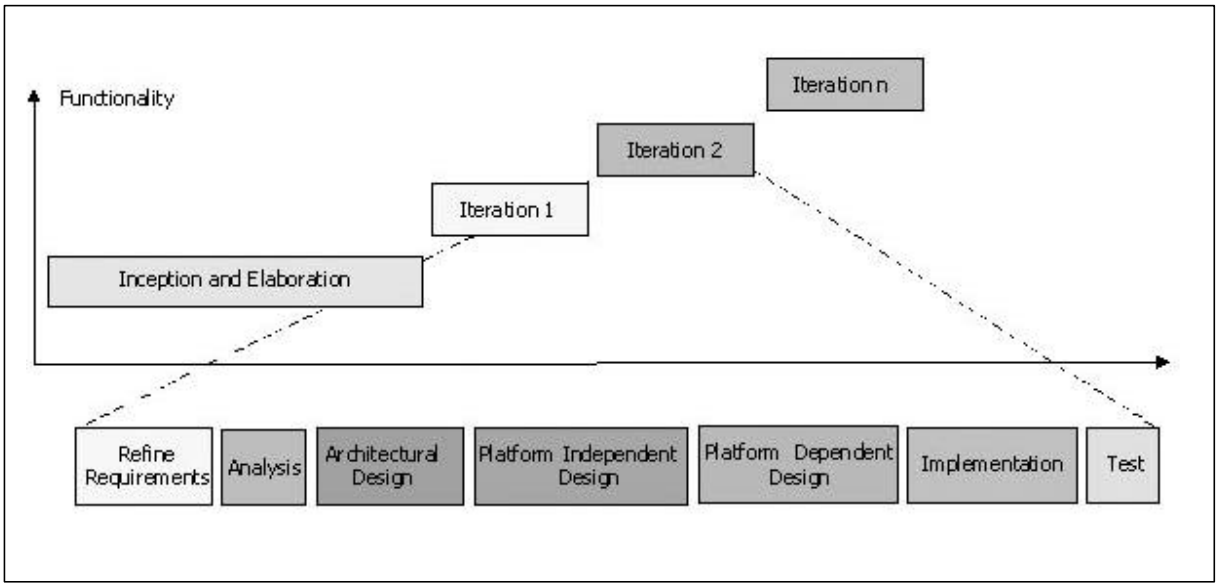

Figure 1: Mobile Agent-based Development Model.

The original contribution of this paper is to present a more complete methodology of developing mobile-agent based applications that combines and extends ideas presented in previous works, according to widely used software engineering practices. Current approaches to develop agent-based software do not satisfactory explores mobility.

The main disciplines of the model shown in Figure 1 are briefly described. For a more complete presentation, see [8]. The aim of the **inception and elaboration phase**, as in the unified process, is to delimit the scope of the project, investigate problem domain and identify functional and non-functional requirements. In order to guarantee that the requirements are acquired without directing to specific solutions, it is strongly advisable not to consider mobile agent concepts. Typical artifacts produced are: software development plan, business case, requirements specification, glossary, systems architecture, preliminary conceptual model and prototypes (usually optional). The **construction phase** consists of analysis, design, implementation and test disciplines. They are presented in the next sections. As the project reaches initial operational capability, it enters the **transition phase** as in the conventional unified process.

2.1 Analysis

The main objective of this discipline is to produce a high-level model of the problem domain that can be used by both stakeholders and designers according to the scope of the current iteration. Documents such as business case, functional and non-functional requirements, use case diagrams and project plan are common entries. The main tasks to be performed involve detailing use cases (basic and alternate flows, preconditions and postconditions), creating activity diagrams, a conceptual model (class diagram representing domain concepts) and a glossary as well as validating them. The behavior of the system can be more precisely specified, for instance, by system sequence diagrams and by system operation contracts [14]. These activities are executed as usual in our methodology, expect from the fact that we must take special care when dealing with agents and mobility.

2.1.1 Detailing Use Cases

A detailed specification of use cases usually includes: name, actors, description, basic and alternate flows, preconditions and postconditions [14].

2.1.2 Defining a Conceptual Model

A conceptual model (also called domain model) illustrates concepts in the problem domain without referring to particular solutions. It is desirable for this model to be generic enough to allow for different solutions to be chosen. For instance, if concepts related to mobile agents are unnecessarily used early in the development process, there is a risk of limiting the number of possible solutions to the ones involving mostly mobile agents. So, we recommend avoiding using them at analysis level.

Autonomous entities are the ones that are responsible for performing a particular task and indeed exist in the problem domain. The objective of modeling them at analysis level is to capture autonomy and mobility in the problem domain whenever they really exist and appropriately document them. It is important to remark that there should be no compromise to implement this entity as a mobile agent and it should not limit the number of possible solutions. On the contrary, this should lead to a bigger number of feasible possible solutions – also including the ones that involve mobile agents. Experience has shown that the gap between conventional object-oriented analysis and mobile agent-based design can be too large, since the former is based mostly on sequence and message passing, not properly capturing other aspects of the system [3,19,20]. This is a controversial matter that is out of the scope of this paper.

2.1.3 Specifying Contracts for System Operations

Contracts describe the outcome of executing a system operation in terms of state changes to domain objects [14]. Use cases are generally used to describe system behavior in the unified process. However, in some cases, it can be of great value to detail information from particular operations that are part of a use case. Contracts are usually created, at the level of detailed analysis or high-level design, for each system operation that emerge when sequence diagrams are created for expressing the behavior of use cases. They can be seen as artifacts in the intermediate level of analysis and design. At design level, concrete concepts such as mobile agents are used.

A contract may include the following information: operation name, arguments and responsibility, exceptional conditions, output, preconditions and postconditions. Preconditions are noteworthy assumptions about the state of the system or objects in the conceptual model. Postconditions represent the state of objects in the conceptual model after completion of the operation. Larman [14] has categorized postconditions as instance creation and deletion, attribute modification and associations formed and broken (UML links). Concerning mobile agents, we suggest that further categories be considered at design level: agent migration, agent cloning, agent deactivation and agent activation.

2.2 Design

In this discipline, analysis models are extended and refined to provide a feasible solution to the system, considering functional and non-functional requirements. Due to particularities of mobile agents technology, in the model presented in this paper, this discipline is divide into three sub-disciplines: platform independent architectural design, platform independent detailed design and platform dependent detailed design. This is based on the ideas presented by Tahara et al [22]. There, agent-based systems are designed according to the following levels: macroarchitecture – high-level architectural design independent of platforms – and microarchitecture – detailed design and agent behavior specialized in each platform to be considered. In our model, the architectural design and platform dependent detailed design disciplines corresponds to the macroarchitecture and microarchitecture design levels proposed by Tahara et al. The main difference between the two models is that we included a platform independent detailed design discipline where an extension of UML is used to model the detailed logical design of the system, independently of particular platforms. Also, we consider a slightly different classification of patterns as platform independent or dependent as in [13]. As mentioned before, this division is motivated to promote design reuse and also because mobile agent-based applications may be required to run on different platforms. The disciplines of design are presented in the next sections.

A mobile agent has the unique abilities to transport itself (code and state of execution) from one system in a network to another and also create a clone of its own whose execution begins at the point of the cloning. An agent system, or simply *agency*, is a running agent platform that can create, interpret, execute, transfer and terminate agents. This is usually associated with the authority of a person or organization. To move from one agency to the other, agents must have special permissions.

2.2.1 Architectural Design

Architectural design represents an outline of the system. Particularly, in a mobile agent-based development, it

presents agent behavior patterns. When applying these patterns, it is important to take functional and non-functional requirements into account in order to guarantee they are fulfilled. The main tasks to be performed are: decomposing the system into layers and partitions (conventional layered or partition architecture) and/or modeling agent behavior for the main scenarios according to specific architectural design patterns (behavior diagram).

Architectural, or macroarchitectural, mobile agent behavior patterns are usually classified as: mobility and task patterns [13,22]. Mobility patterns are applied when agents perform a task while moving in a chain of agencies. The intention is to allow for an efficient use of network resources. On one hand, it is advantageous to access resources locally. On the other hand, costs of moving agents must be taken into account. Moreover, encapsulation of mobility management is usually addressed in order to improve the quality of design. Task patterns are used to specify how agents should interact in order to perform a task. In general, tasks can be dynamically assigned to agents. Also, a given task can be performed either by a single agent or by a group of agents working in parallel and cooperating.

A behavior diagram presents a high level view of the combination of agent behavior patterns used to design the system according to specific scenarios. For the sake of simplicity, we use an informal notation to present such diagrams in Section 3. However, we recommend the use of an established notation to describe the architecture of interconnected systems such as UML-RT [21]. Behavior diagrams can be composed of blocks that represent hosts and servers and arrows that represent client-server relations. Each host may hold a number of agencies that are represented by rectangles. Agents that run on specific agencies are included in the rectangles. The blocks are connected by dashed arrows to represent agent mobility, single arrows to represent agent creation and double arrows to represent agent cloning.

2.2.2 Platform Independent Detailed Design

The objective is to produce a specification of a logical design independently of any specific mobile agent platform. Artifacts must reflect how agent behavior is implemented according to the architectural patterns chosen and the functional and non-functional requirements. The main tasks involved in this discipline are to refine class diagrams to include implementation concepts (mainly from patterns chosen) and elaborate interaction diagrams to show how operation contracts are implemented and fulfill postconditions (see Section 2.1 for contracts).

Regarding class diagrams, agents are usually modeled as classes. However, when specifying an agent's class, it is important to decide for a minimum set of data that should migrate with the agent, since the size of an agent may have a considerable impact on the performance of the whole system.

Regarding interaction diagrams, they are used as usual to illustrate sequences of interactions of the system. Two kinds of interaction diagrams can be created: sequence and collaboration. For these diagrams, we use an extension of UML proposed in [12] for modeling mobile agent-based applications. This extension introduces new elements to represent agencies, regions, agents, migration and cloning.

2.2.3 Platform Dependent Detailed Design

The main objective is to extend the logical specification produced in the platform independent design in order to address design and implementation issues related to a specific agent system platform. In this paper, we illustrate this discipline focusing on the Grasshopper platform [10]. As one would expect, complete and consistent artifacts of analysis, architectural project and independent platform design are required as input criteria. The main tasks to be performed are the following: apply platform design patterns and refine interaction and class diagrams to reflect these patterns. Platform dependent patterns are usually categorized as: (1) refinement of patterns to include details related to implementing them in the platform and (2) patterns related to issues such as communication, security and safety. Regarding output criteria, it is important to check completeness and consistency of diagrams with respect to input artifacts. For the lack of space, we briefly comment on mobility and communication in the platform.

Mobility. Mobile agent platforms can support two forms of mobility: strong mobility and weak mobility [4]. Strong mobility is the ability of an agent to migrate with both code and execution state. Weak mobility is the ability of an agent to transfer code. In this case, code may be accompanied by some initialization data, but no migration of execution state is involved. Due to JVM constraints (e.g. execution state of threads cannot be captured), Grasshopper as well as the great majority of other platforms based on Java, does not support strong mobility. It supports only weak mobility. Therefore, programmers need to create a mechanism to simulate strong mobility. A Grasshopper agent must implement a method named *live()*. This method defines the agent behavior, i.e., the flow of control executed by its thread. In order to simulate strong mobility, this method can be split in execution blocks by using conditional commands. Each block covers operations that have to be executed in a given (type of) location. A block finishes its execution by transferring the agent to another location.

Communication. The Grasshopper platform offers communication services that can be used according to specific communication patterns. One desirable property of agent systems is to allow for transparent locality. In other words, agents need not care about the location of the desired communication peer. This is implemented in Grasshopper by a communication pattern named *Proxy*. Proxies are entities that intermediate communication between client and server and are responsible for establishing the required connection. When a server is a mobile agent, the proxy is responsible for finding out where the agent is located. A client accesses a proxy by a local call. Then, the proxy forwards the call via the communication channel to the server.

2.3 Implementation and Tests

Having a platform dependent detailed design, implementation consists basically of coding the necessary classes. Tests can follow a process composed of the following activities: planning, specification, construction, execution and analysis of results [15]. These activities can be integrated with the analysis and design disciplines. We recommend that unit testing, integration testing and functional testing as well as system testing to be performed. However, it is very important to verify non-functional and structural aspects related to mobility, communication and security.

3 Case Study: Conference System

The case study presented in this paper, suggested by Cardelli [3], is about a conference reviewing system to manage a virtual program committee meeting for a conference, including conference announcement, paper submission, assignments to committee members and generation of review forms, paper review, report generation, conflict resolution, notification, final versions and publication. It is important to remark that we follow the requirements documented in [3].

As a requirement, most interactions must happen in absence of connectivity. For instance, it is highly unlikely that all committee members will be continuously connected during review and conflict resolution. Also, referees do not need to be continuously connected to fill forms that may require semantic checking. Review forms may be forwarded to a chain of referees and must find their way back to the committee member that is responsible for it. Moreover, the program chair must be freed from the task of following all operations involved in forwarding forms and getting them back. The same applies to reviewers. Another requirement is that the system must handle multiple administrative domains, since referees are geographically dispersed. Finally and not less important, to fully automate all activities, forms must be active. This may guarantee that error-prone tasks such as collecting, checking and collating pieces of information as well as distributing them are done efficiently. In this paper, we focus on generating and distributing review forms and paper review.

For the lack of space, only a few artifacts of analysis and design are presented in a summarized way. A complete presentation of this case study is given in [8]. Other case studies can be found in [17,16].

3.1 Analysis

In the following sections, the main activities and artifacts of analysis for the case study are described.

3.1.1 Detailing Use Cases

After paper submission, the program chair collects submission forms together with attached papers and make the necessary preparations for the reviewing process to begin. We defined 3 separate use cases for this as follows. Note that each committee member is also a reviewer. He/she may decide to review the paper directly, or to send it to another reviewer.

Generate Form The program chair (actor) asks for the review process of a given paper to be started. For this, a review form is created and a committee member is assigned to be responsible for the reviewing of the paper. The form is sent to the committee member who is responsible for forwarding it to N reviewers, where N is the number of reviewers required. Review forms contain the paper and information on how to come back to the appropriate committee member/reviewer.

Forward Form A reviewer (actor) receives a review form, opens it and decides to send it to another reviewer. He may ask that the form come back to him once the review is completed. The intention is that each reviewer can check the work of the subreviewers. As an alternate flow, the reviewer receives a form, opens it and postpone his decision

of either reviewing the paper or redirecting it.

Review Paper A reviewer (actor) receives a form, opens it and decides to review the paper. After reading the paper, the reviewer fills in the form fields and releases it. The form returns, by default, to the program chair, unless an intermediate reviewer wishes to check the review. As an alternate flow, the reviewer may decide to complete the review latter.

3.1.2 Defining a Conceptual Model

A high-level conceptual model of the conference system is presented in Figure 2, representing the main concepts and their relationships.

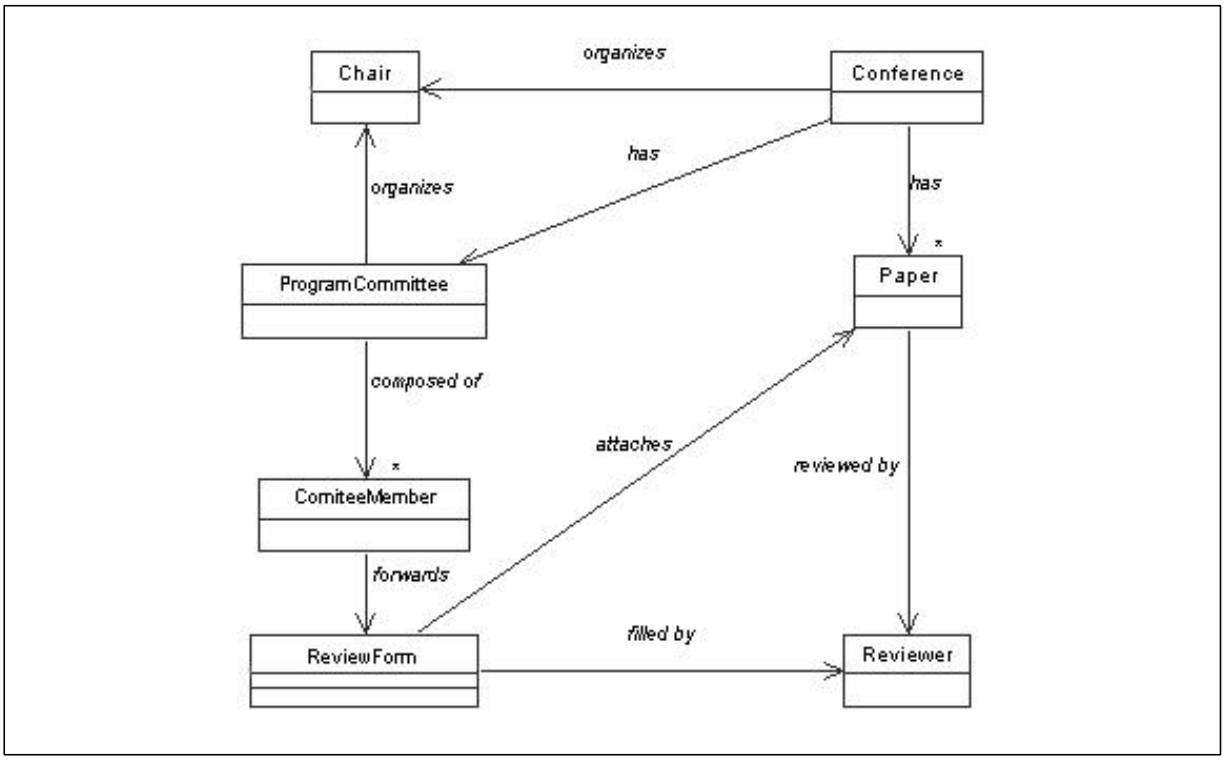

Figure 2: Conference System: Conceptual Model.

3.1.3 Specifying Contracts for System Operations

The **Generate Form** use case gives rise to a system operation named *generateForm()*. A brief description of a contract for this operation is given as follows. This operation is responsible for creating a form review (*ReviewForm*) and delivers it to a committee member, responsible for forwarding it to *N* reviewers. As preconditions, (1) a valid paper submission must exist, (2) the target committee member identification must be valid, and (3) the number of reviewers must be a valid positive and non-zero integer *N*. As postconditions, (1) a *ReviewForm* is created, (2) the review form is sent to the committee member, and (3) the committee member creates *N*-1 copies of the form. As exceptional conditions, the committee member may not be ready to receive the *ReviewForm*. In this case, the system must try to send the form again after a certain period of time. And, after a certain number of trials, an error must be reported.

3.2 Design

In this section, a mobile agent-based design to implement the case study is presented. A mobile agent named *ReviewFormAgent* is created to model an entity responsible for moving a *ReviewForm* through the chain of reviewers. A stationary agent named *ChairAgent* is created to act in the behalf of the program chair and freed he/she

from the task of controlling the *ReviewFormAgent* agents.

3.2.1 Architectural Design

In Figure 3, an agent behavior diagram illustrates the review process. A combination of mobility patterns called *Itinerary* and *Branching* is used [13]. Also, cooperation is defined by the master-slave task pattern, where *ChairAgent* is the master and *ReviewFormAgent* agents are slaves. *ChairAgent* is responsible for creating an agent– *ReviewFormAgent* – to conduct the review of each article and get back the results when the review finishes (the slave is back). *ReviewFormAgent* is defined as a mobile agent that migrates from one agency to the other to perform the reviewing task, according to an itinerary that is defined in the reviewing process. This agent can be forwarded as appropriated from one referee to the other and back to a program committee member. Program committee members are responsible for distributing an article to a number of reviewers. Clones of the review form are sent to different reviewers (branching). Business objects are kept in a database server and agent *codebase* is kept in a HTTP server.

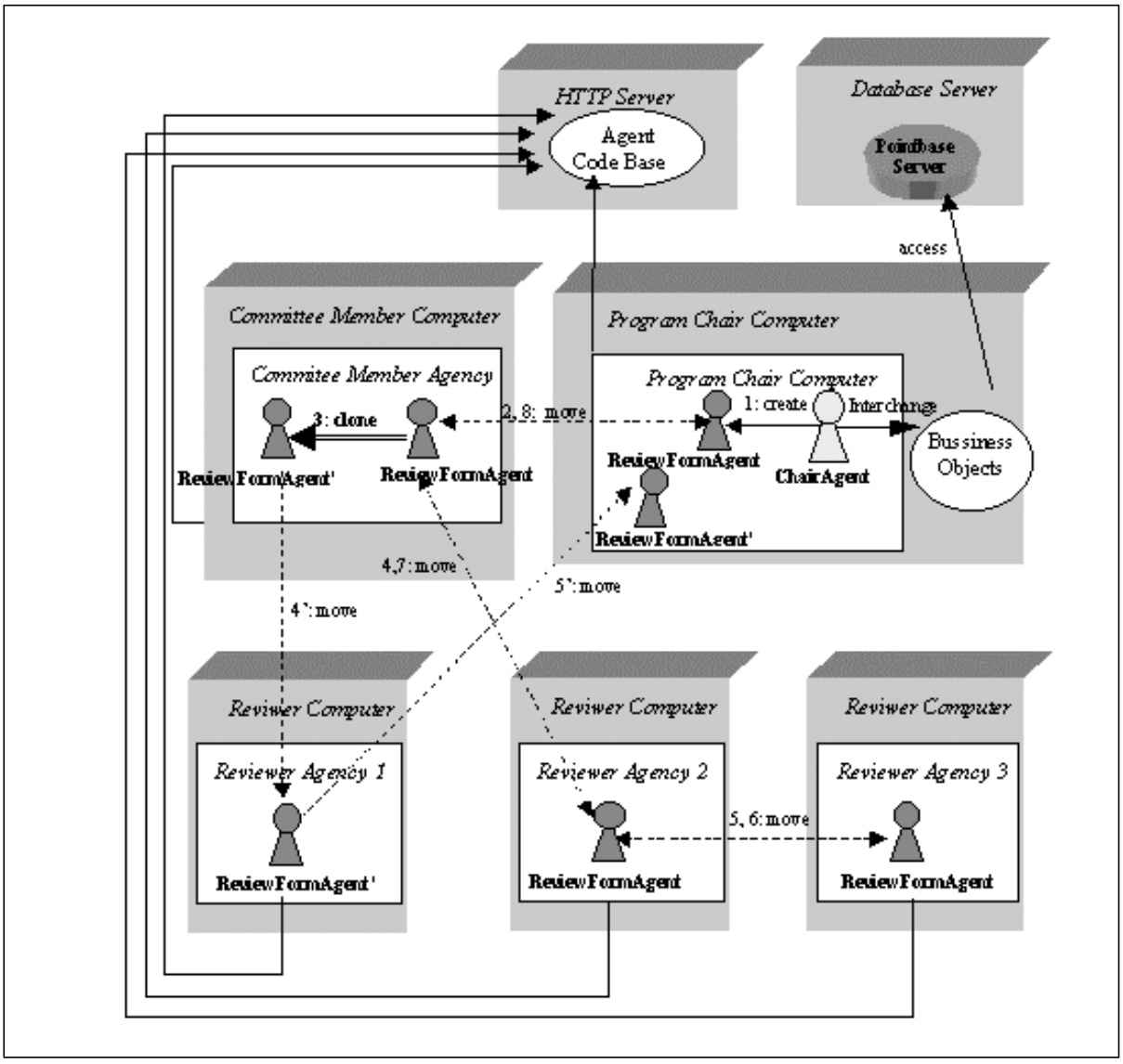

Figure 3: Conference System: Agent Behavior Diagram.

3.2.2 Platform Independent Detailed Design

Figure 4 shows the final part of a sequence diagram for the *generateForm()* operation, considering the patterns chosen. The program chair actor requests to the *ChairAgent* that a review form be created for a given paper and be sent to a given committee member. For this, *ChairAgent* interacts with an external object *Conference* to collect data

about the conference and also with its agency to request creation of a *ReviewFormAgent*. This agent creates its itinerary and migrates to the committee member agency.

Mobility and task patterns are detailed as follows.

Mobility patterns. Encapsulation of mobility management is usually addressed in order to improve the quality of design. As previously mentioned, in our case study, a variation of the *Itinerary* pattern is used. An object *Itinerary* is responsible for keeping information about next destinations. Being an autonomous mobile entity, an agent is capable of navigating itself independently to multiple hosts. In this case, it should be able to handle exceptions such as unknown hosts while trying to be dispatched to new destinations. It might even need to modify its itinerary dynamically. Therefore, the idea is to shift the responsibility for navigation from the agent object to an Itinerary object. The itinerary class will provide an interface to maintain the agent itinerary and to dispatch it to new destinations [13].

The itinerary pattern has been used to represent *ReviewFormAgent* destinations. Actually, an itinerary is built as this agent navigates to represent the path it has to follow on his way back to the program chair host. This is due to the requirement that reviewers may require to check subreviewers work as presented in Section 3.1. From Figure 3, a *ReviewFormAgent* agent migrates to a committee member's agency and clone itself according to the number of reviews required. Committee members interact with this agent to forward them to reviewers. Usually, committee members require the agent to visit his agency on its way back to the program chair agency. As mentioned before, reviewers can either review the paper or forward it to another reviewer.

Task patterns. In our case study, we opt for a pattern called *Master-Slave*. In this pattern, a master agent creates slaves that migrate to remote agencies in other to perform a task. Slaves come back to the master agency with the results produced. This explains why we decided to model the *Chair* agent (master) as a stationary agent. *ReviewFormAgent* mobile agents are slaves (see Figure 3).

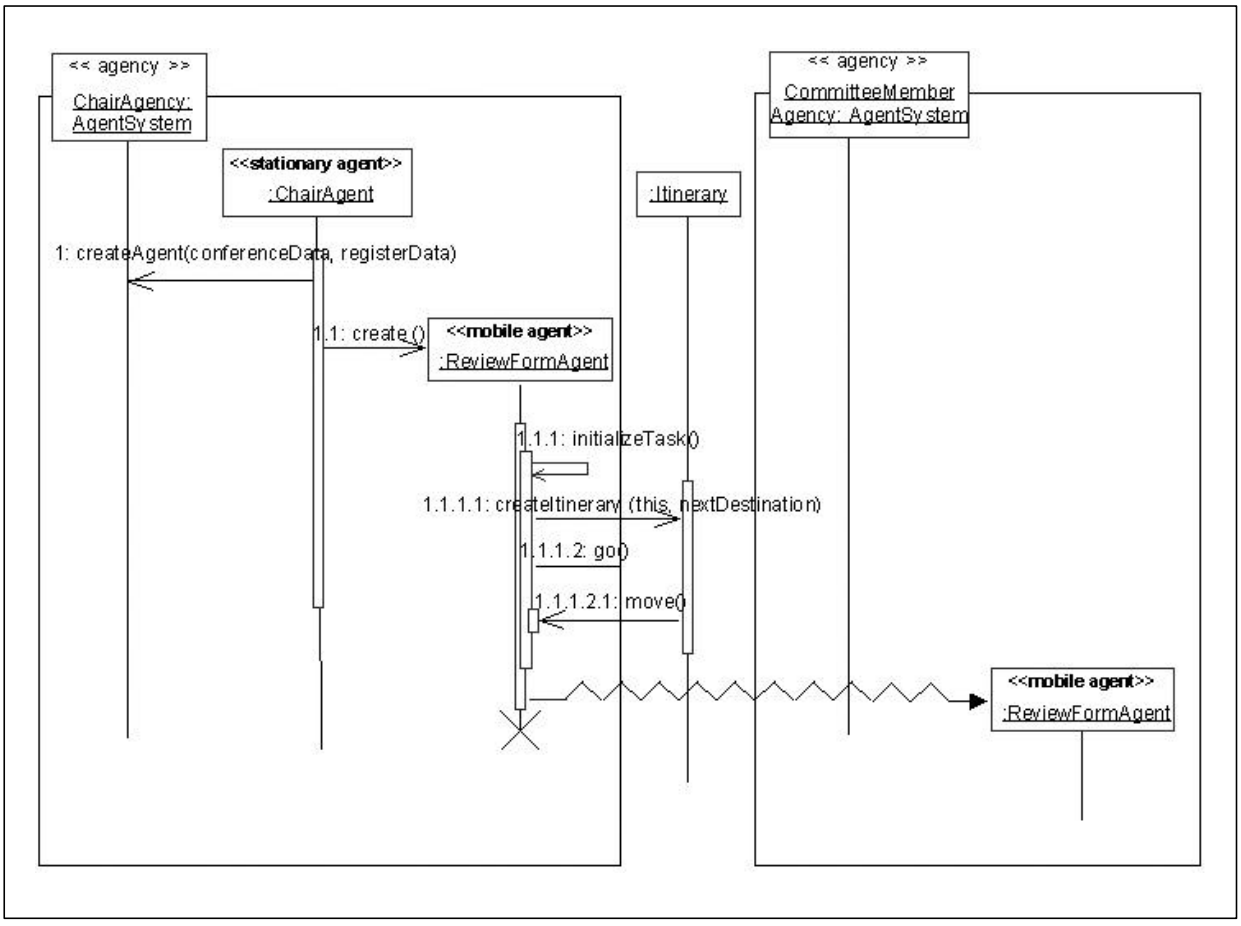

Figure 4: Conference System - *generateForm()* operation - Platform independent sequence diagram.

3.2.3 Platform Dependent Detailed Design

In our case study, based on the Grasshopper platform, *ChairAgent* has to interact with *Conference* in order to get information about the conference that the *ReviewFormAgent* needs to carry. *ChairAgent* has also to interact with its agency to request an agent *ReviewFormAgent* to be created. Due to the use of the proxy pattern, new classes and associations emerge in the project. Sequence diagrams produced in the platform independent design need to be expanded to include proxy objects. An extension of part of the diagram in Figure 4 is presented in Figure 5 (see *AgencyProxy*).

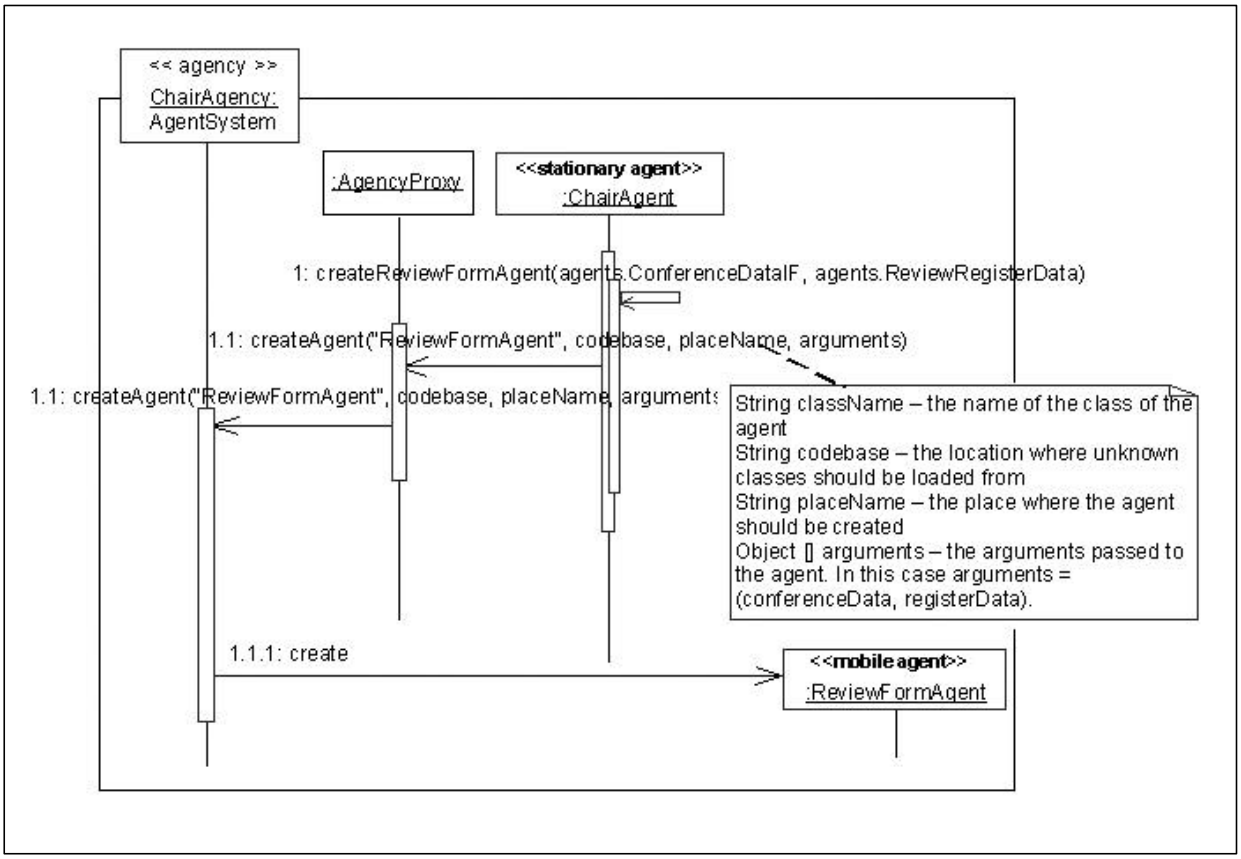

Figure 5: Conference System - *generateForm()* operation - platform dependent sequence diagram.

4 Concluding Remarks

We present a methodology of developing distributed applications based on the mobile agent-based paradigm. This is based on the widely used unified process, the use of UML and design patterns and also techniques and notations for developing mobile agent-based systems. It is clear that, in order to promote the use of mobile agent-based concepts and technology in practice, it is essential to have a well-defined methodology according to a software engineering discipline together with case studies that illustrate its application. Therefore, the main contribution of this work is to combine and extend ideas presented by previous works on techniques and notations for different stages of developing mobile agent-based systems and give additional guidelines on how to proceed from analysis to detailed design. This is illustrated by a case study. It is been widely accepted that current object-oriented methodologies are not appropriate for developing agent-based systems.

The case study presented in this paper is implemented in the Grasshopper platform. Along with communication support, the platform also provides for safety and security [10]. Functional tests have been performed to illustrate the main scenarios described by use cases using both local and remote hosts (under Windows/Linux). The model has also been used in a teaching course on advanced telecommunication systems development. Other two major case studies have also been produced using the model: "Monitoring changing conditions on a flight reservation/selling system" and "Interoperability of distributed commercial corporations".

As further work, the model needs to be completed and thoroughly validated. Disciplines such as Test have not

been properly addressed yet. We have mostly applied functional, unit and system testing as usual. Nevertheless, there is still little knowledge about how such applications, and even Internet-based applications, can be effectively tested. We also need to consider what metrics can be used to manage and evaluate the methodology and the quality of its artifacts. Tool support has also to be considered. We have used Rational Suite Enterprise (http: //www.rational.com) for producing most of analysis and design artifacts. However, additional tools may be required, particularly regarding UML extensions, test techniques, reverse engineering and code generation and documentation. The use of formal notations to specify contracts and patterns may allow for automatic functional tests generation and precise analysis of design properties.

References

[1] M. C. Bernardes and E. S. Moreira. Implementation of an intrusion detection system based o n mobile agents. In *International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000)*. IEEE, 2000.

[2] G. Cabri, L. Leonardi, and F. Zambonelli. Engineering mobile-agent applications via context-dependent coordination. In *Proceedings of International Conference on Software Engineering - ICSE'2001*, 2001.

[3] L. Cardelli. Abstractions for mobile computation. In *Proceedings of Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*, 1999.

[4] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24, 1998.

[5] A. F. Garcia, V. T. Silva, C. J. P. Lucena, and R. L. Milidiú. An aspect based object oriented model for multi-agent systems. In *XVI Brazilian Symposium on Software Engineering*, 2001.

[6] R. H. Glitho, E. Olougouna, and S. Pierre. Mobile agents and their use for information retrieval: A brief overview and an elaborate case study. *IEEE Network*, 2002.

[7] The Object Management Group. *The Mobile Agent System Interoperability Facility*. The Object Management Group, Framingham, MA, 1997.

[8] F. P. Guedes. Um modelo para o desenvolvimento de aplicações baseadas em agentes móveis. Master's thesis, Pós-Graduação em Informática, Universidade Federal de Campina Grande, 2002. http: //www.dsc.ufcg.edu.br/~copin/ (banco de dissertações).

[9] Hayzelden and Bigham, editors. *Agents for Future Communication Systems*. Springer, 1999.

[10] IKV++, Germany. *Grasshopper Basics and Concepts*. http: //www.grasshopper.de.

[11] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[12] Cornel Klein, Andreas Rausch, Marc Sihling, and Zhaojun Wen. Extension of the Unified Modeling Language for mobile agents. In Keng Siau and Terry Halpin, editors, *Unified Modeling Language: Systems Analysis, Design and Development Issues*, chapter 8, pages 116–128. Idea Publishing Group, 2001.

[13] B. D. Lang and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley, 1998.

[14] C. Larman. *Applying UML and Patterns - An Introduction to*. Prentice Hall PTR, second edition, 2002.

[15] J. D. McGregor and D. A. Sykes. *A Practical Guide to Testing Object-Oriented Software*. Addison-Wesley, 2001.

[16] P. S. Medcraft, C. S. Baptista, and U. Schiel. DIA: Data integration using agents. In *ICEIS'2003 - 5th International Conference on Enterprise Information Systems*. ICEIS Press, 2003.

[17] V. N. Medeiros. Desenvolvimento e verificação de aplicações baseadas em agentes móveis. Technical report, DSC/Universidade Federal de Campina Grande, 2002. Relatório de Conclusão de Projeto de Pesquisa, PIBIC/CNPq.

[18] D. Milojicic, F. Douglis, and R. Wheeler, editors. *Mobility: process, computers and agents*. ACM, 1999.

[19] J. Mylopoulos, M. Kolp, and J. Castro. Uml for agent-oriented software development: The tropos proposal. In M. Gogolla and C. Kobryn, editors, *Proceedings of UML 2001*, volume 2185 of *Lecture Notes in Computer Science*, pages 422–441, 2001.

[20] J. Odell, V. D. Parunak, and B. Bauer. Extending uml for agents. In *Proceedings of the Agent-Oriented Information System Workshop at the 17th National Conference on Artificial Intelligence*, pages 3–17, Austin,

USA, July 2000.

[21] B. Selic and J. Rumbaugh. Using uml for modelling complex real-time systems. Technical report, Whitepaper, 03 1998.

[22] Y. Tahara, A. Ohsuga, and S. Honiden. Agent system development method based on agent patterns. In *Proceedings of International Conference on Software Engineering - ICSE'99*, 1999.

[23] Venieris, Zizza, and Magedanz, editors. *Object Oriented Software Technologies in Telecomunications*. John Wiley & Sons, 2000.

[24] M. Wooldridge, N. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents - Agents'99*, 1999.

[25] M.-J. Yoo, J-.P. Briot, and J. Ferber. Using components for modeling intelligent and collaborative mobile agents. In *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 1998*. IEEE, 1998.