FROM m-GAIA TO GRASSHOPPER: ENGINEERING MOBILE AGENT APPLICATIONS

W. Sutandiyo, M. B. Chhetri, S. Krishnaswamy, S. W. Loke
School of Computer Science and Engineering
Monash University

Abstract

There is a need for agent oriented software engineering (AOSE) methodologies that support the conceptual modelling of mobile agent systems. In this paper, we present m-GAIA, our extension to the GAIA methodology for modelling mobile agent systems. m-GAIA incorporates explicit constructs to perform the analysis and design of multiagent systems which include mobile agents. We also present our experiences in mapping the conceptual models developed in m-GAIA to an implementation using the Grasshopper mobile agent toolkit.

1. Introduction

A property of agents is the ability to move from one host to another to perform computations. While not all agents require such a property, agent mobility has been recognized as beneficial in a number of applications, including information-centric applications [3] and a sizeable number of mobile agent toolkits (http://mole.informatik.uni-stuttgart.de/mal/preview/preview.html) have been developed for mobile multiagent applications. While agent mobility may not be needed in all multiagent systems, we contend that it can be disadvantageous if it is a concept only confined to (and considered in) the implementation phase, or as an after-thought in engineering multiagent systems (e.g., used for optimising applications). The advantages of agent mobility might remain unexploited, adding mobility to agents after the system implementation can be problematic, or development of multiagent systems where some agents can be mobile might proceed in an ad hoc manner, if agent mobility is not considered in the earlier phases of agent-oriented software development. Whether agents are mobile or stationary does have an impact on the architecture and required components of a multiagent system, and how they can be made so, can be an important factor in deciding which agent toolkit to use for implementation. The why and how of agent mobility, or its dismissal, should be considered, and if needed, its rationale documented at design time. Such consideration of agent mobility should also support toolkit independence (as far as the required functionality will allow). On the other hand, there will some applications where agent mobility is obviously useful and more efficient, but an implementation in some mobile agent toolkit without the auspices of an integrated agent-oriented analysis and design is clearly less than ideal. What is the current state of the art in methodologies for the engineering of multiagent systems where some agents are mobile? Several approaches (or methodologies) for addressing the analysis and design phases of agent-oriented software have been reviewed [7]. Our analysis of these

approaches shows that there has been little focus on supporting the conceptual modelling needs of mobile agent systems. The two approaches that do support analysis and design of mobile agent systems include MaSE [6] and some Petri Net based techniques [4]. MaSE includes a *move* activity in its analysis phase, and in the design phase, *mobile components* that allow specification of the activities that result from the *move* operation. While focussing on only one aspect of modelling mobile agent systems. MaSE ignores other aspects such as distinguishing conceptually between agents that can be mobile and those that cannot, and modelling the concepts of *locations*, *migration rationale* and *itineraries* for mobile agents. Moreover, MaSE extends the object-oriented approach rather than start with a "pure" multiagent background [7]. The Petri Net approaches typically model only an aspect of the agents (e.g. itineraries in [4]) and is not as comprehensive as methodologies such as GAIA [8].

We note that while multiagent systems constructed using the above methodologies might be mobile, we say that they lack support for modelling mobile agent systems as the concept of agent mobility is not explicit in the analysis and design phases, and has not been given adequate attention. In our work, we have extended the GAIA [8] methodology for conceptual modelling of multiagent systems to support the analysis and design of mobile agent systems. In this paper, we present m-GAIA, our extension to the GAIA methodology and illustrate its applicability in supporting the analysis and design of a mobile agent application, namely, a smart lecture theatre system. We also present our experiences in mapping the analysis and design specifications developed using m-GAIA to the Grasshopper™ (http://www.grasshopper.de) [1] toolkit for implementing mobile agent systems. The paper is organised as follows. In section 2, we present m-GAIA. Section 3 outlines the architecture and operation of the smart lecture theatre system and presents the implementation of the smart lecture theatre system using the Grasshopper toolkit. It maps the m-GAIA conceptual methodology to a mobile agent implementation environment. Section 4 concludes the paper.

2. m-GAIA

We present m-GAIA as an enhancement of GAIA [8] to facilitate conceptual modelling of mobile agent systems. The GAIA methodology allows the software developer to analyse and design the system after the requirements are collected [2] [5] and move from abstract (analysis) to concrete (design) level of agent systems. *Figure 1* without the *mobility model* illustrates the GAIA methodology structure. It consists of the *analysis* and *design* phases. The objective of the analysis phase is to obtain an understanding of the system and its structure. The analysis phase consists of the *roles model* and the *interaction model*. The roles model identifies the roles in the system and the interaction model identifies the interactions between the roles found. There are four attributes of roles: responsibilities, permissions, activities, and protocols.

- Responsibilities consist of two properties: liveness property defines the continual execution of the role within the system; safety property is a condition that must be maintained to avoid system behaviour that is contrary to its system requirements.
- Permissions define the access privileges or rights of roles.
- Activities are tasks that need to be performed by roles without interaction with other roles.
- Protocols are activities that involve interactions with other roles.

The objectives of the design phase are to convert the system from an abstract level to a concrete level and to ease implementation. The design phase consists of the *agent model*, the *services model*, and the *acquaintance model*. The agent model is used to map the roles to agent types. The services

model lists the services that each role can provide and be associated with. The acquaintance model defines the relationships between agents.

Our rationale for choosing GAIA was its modularity and simplicity, which provide scope for extensioning while retaining consistancy in notation. In order to support conceptual modelling of mobile multiagent systems, m-GAIA incorporates the existing models of GAIA and adds a new model, namely, the *mobility model*. The following section describes m-GAIA. The basic ideas of m-GAIA are borrowed from the existing GAIA methodology. As such, m-GAIA still consists of two phases, which are the analysis and design phases. The analysis phase includes the roles model and the interaction model. The design phase includes the agent model, the acquaintance model, the services model, and the *mobility model*. The structure of m-GAIA's models is illustrated in *Figure 1* below.

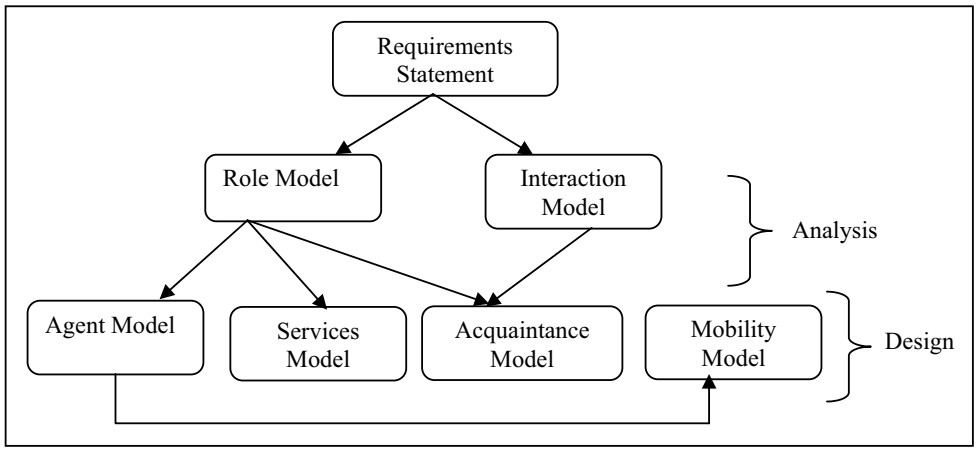

Figure 1: Structure of m-GAIA's Models

In order to help developers analyse and design the entire multiagent systems (including mobile agents), each model in m-GAIA has particular features that are consistent with the GAIA methodology. The roles model of m-GAIA identifies the roles to take into account within the entire system. Besides identifying the roles, the roles model also includes protocols. The protocols are activities that a role encounters within the system lifecycle and it involves interaction with other agents. The interaction model of m-GAIA defines the interactions between protocols with other roles. The agent model of m-GAIA is used to identify the agent types and how many agents are involved within the entire system. However, unlike in GAIA, in m-GAIA we include constructs to distinguish between agents that possess the characteristic of mobility and those that do not. The services model of m-GAIA is the list of services that each role can provide and be associated with. The acquaintance model defines the communication links between each agent. The mobility model of m-GAIA defines the mobility characteristics of agents further, such as identifying the movements and travel path of each mobile agent. m-GAIA's models still serve the same purpose as the corresponding models in GAIA.

Like GAIA, m-GAIA has abstract concepts and concrete concepts. The abstract entities are entities used during the analysis process and they do not necessarily have direct correlations in the run-time system. The concrete entities are entities that are considered during the design process. The concrete entities have direct correlations in implementation of the run-time system. *Table 1* summarises the abstract and concrete concepts of m-GAIA.

Abstract Concepts	Concrete Concepts
Roles	Agent types
Role types	Services
Permissions	Acquaintances
Responsibilities	*Place types*
Protocols	*Atomic movement*
Activities	*Travel paths*
Liveness properties	
Safety properties	

Table.1: Abstract and Concrete Concepts in m-GAIA

It must be noted that in *Table 1*, the italicised concepts are unique to m-GAIA and mainly aim to support modelling agent mobility in multiagent system. The additional features involve modifications to two of the existing GAIA's models, which occur in the roles model and the agent model: (1) In the roles model, the roles identified are categorized into three distinct *role types*, which are system, interface, and user roles. (2) In the agent model, the agents are categorised into mobile or stationary. In addition, m-GAIA has the mobility model, which GAIA does not. Steps in building the mobility model and its sub components will be discussed further in section 2.2. The interaction model, the services model, and the acquaintance model are imported from GAIA into m-GAIA without change.

2.1. Analysis Phase of m-GAIA

Like the GAIA methodology, the analysis phase of m-GAIA consists of the roles model and the interaction model. However, in m-GAIA, modifications have been made to roles model. The following section will discuss the role schema and the modification which has been made.

2.1.1. The Roles Model

The roles model of m-GAIA aims to identify the roles within the entire system. Each role identified is categorized into three different role types - *system*, *interface*, and *user* roles. The purpose of categorising roles is to clarify each role's responsibilities within the system. A system role is defined as a role that interacts with other parts of the system and not the user. An interface role is a role that interacts with the user and the other parts of the system. A user role is a role that represents the human user itself. Despite the modification in the roles model, the remaining components are the same as in GAIA. The roles model of m-GAIA is illustrated in *Figure 2*.

Role Schema: *name of role –**role type***

Description:
Short English description of the role

Protocol and Activities:
Protocols and activities in which the role plays a part

Permissions:
"rights" associated with role

Responsibilities:
Liveness: *Liveness responsibilities*
Safety: *Safety responsibilities*

Figure 2: Template for role schemata in m-GAIA

2.2. Design Phase of m-GAIA

The design phase of m-GAIA consists of the agent model, the acquaintance model, the services model, and an additional model called the mobility model. We modified GAIA's agent model to specify the mobility characteristic of agents. The acquaintance model and the services model are the same as those in GAIA. The following section will discuss the agent model and the modifications which have been made.

2.2.1. The Agent Model

The agent model is used to identify the number of agents, the agent types, and the relationship between the roles identified (in the role model) and the agent types in the system. Unlike GAIA's agent model, m-GAIA's agent model classifies the agents into two different categories - mobile (by adding a notation of "*m*" sign) and stationary. The categorisation of agent types caters for mobility characteristic of agents. Furthermore, we modify the agent model to allow similar behaviour roles to be grouped into one category. This is notational illustrated by grouping the role names between parentheses as shown in *Figure 3*. This modification is for convenience of presentation. *Figure 3* illustrates the agent model of m-GAIA.

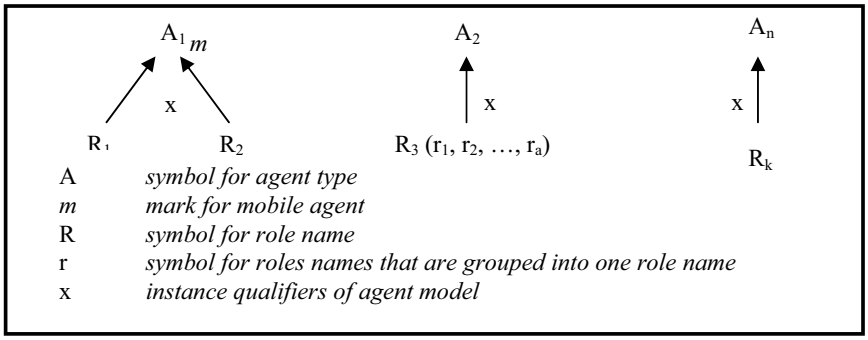

Figure 3: Agent model of m-GAIA

2.1.2. The Mobility Model

The mobility model enhances GAIA to incorporate support for modelling of mobile agents in multiagent systems. The analysis phase of m-GAIA involves identifying the roles and the interactions of each role. Unlike the analysis phase, the design phase of m-GAIA involves agents. Therefore, the mobility model is best fitted into the design phase rather than in the analysis phase, as mobility is a characteristic of agents and not roles. Furthermore, mobility is not an interaction as an agent does not need to be mobile to communicate. These considerations motivated the inclusion of the mobility model in the design phase. The mobility model is derived from the agent model. In the agent model, the agent types are categorised into mobile and stationary. Mobile agents are able to move from one place to another place in order to perform the tasks assigned. Therefore, in order to model the mobility characteristics of mobile agents, the mobility model identifies *place types*. Place types are locations that the mobile agent can visit or reside in. The place types define the working environment of mobile agents. In the Grasshopper [1] mobile agent toolkit, the place type concept is also equivalent to agent's environment called places. The place in the Grasshopper's toolkit is also the mobile agent's execution environment; the mobile agents are able to move from one place to another place. There are four steps in constructing the mobility model:

1. Identify place types.
2. Identify the relationships between agent types and place types

3. Define the cardinality between agent types and place types
4. Identify the travel path of each mobile agent.

Step 1: Place Types

Table 2 illustrates the place types in a mobility model. P_i denotes place types ($i = \{1, 2\ldots n\}$).

Place Types	Description	Instances
P1	*Short English description of places types*	*Instances Operators indicates how many place types exist in the system*
P2		
⋮		
Pn		

Table 2: Place Types

Table 3 defines the instances operators of place types.

Operator	Description
n	There will be exactly n instances
m … n	There will be between m and n instances
*	There will be 0 or more instances
+	There will be 1 or more instances

Table 3: Instances Operators

Step 2: Agents and Places Specifications

Step 2 of the mobility model is derived from step 1 and the agent model. In this step, we identify the relationship between agent types and place types. It also defines the constraints of the relationship. The agents and places specifications are derived from the place types identified in step 1 of mobility model. *Table 4* illustrates the agents and places specifications of mobility model. Let *A* be the symbol for agent types and *m* the number of agent types. We are aware that mobile agents have been defined in the agent model however the ticks (✔) sign to indicate mobile agents in the agents and places specifications are included for the purpose of clarity.

Agent Types	Mobile	Place Types	Constraints
A1	*A tick sign to identify if the specific agent is mobile or non-mobile*	P1, P2	*The constraints of agents and place types relationship*
A2		P3	
⋮		⋮	
Am		Pn	

Table 4: Agents and Places Specifications

Step 3: Cardinality of Agents and Places

The cardinality between agent types and place types shows how many agents of an agent type can reside in a place of a place type. The cardinality of agents and places (step 3) is based on the agents and places specifications (step 2). *Figure 4* illustrates the cardinality of agents and places of the mobility model. The cardinality operators identify the constraint relationship between agent and place types. *Table 3* defines the cardinality operators of agent types and place types.

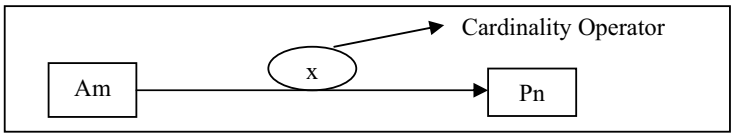

Figure 4: Cardinality of Agents and Places

Step4: Travel Schema of Mobile Agent Types

The travel schema of each mobile agent type includes *origin*, *final destination*, *list of atomic movements*, and *paths*. The origin is the place type where the mobile agent starts the movement to accomplish the tasks assigned. The final destination is the place type where mobile agent will reside after it completed the tasks assigned. The atomic movement is the smallest granularity movement required to accomplish the tasks assigned. The paths are the list of atomic movements that the mobile agent may travel in order to accomplish the tasks assigned. *Figure 5* below illustrates the template for the travel schemata of mobile agents.

Agent Type: *name of agent*

Description: *Short English description the mobility of agent*
Origin: *The origin place type of agent*
Final Destination: *The final destination place type of agent*
List of atomic movements:
Lists of possible movements of agent

Movement ID	*Short English description of the atomic movement*

Paths:
Lists of paths of agent to accomplish each task assigned

Path ID	*List of atomic movements involve in this particular path*

Figure 5: Template for travel schemata of mobile agents

The travel paths of each mobile agent might occur many times in the entire system lifecycle. Therefore the number of times paths are travelled is defined with the counting operators as summarised in *Table 6* below.

Operator	Description
n	There will be exactly n instances
m … n	There will be between m and n instances
*	There will be 0 or more instances
+	There will be 1 or more instances

Table 6: Counting operators

A travel path of a mobile agent is constructed with a combination of the atomic movements of the mobile agent. Therefore, the operators to indicate the composition of atomic movements is summarised in *Table 7*.

Operator	Description
x.y	x followed by y
x | y	x or y occurs
[x]	x is optional

Table 7: Path Operators

3. Mapping m-GAIA Models to a Grasshopper Implementation

In this section we present an application that we term "Smart Lecture Theatre" that includes a combination of mobile and stationary agents. We have performed a design and analysis of this system using m-GAIA and in this section demonstrate how the m-GAIA models map to an implementation of such applications using current mobile agent toolkits. The Smart Lecture Theatre is based on the pervasive computing concept of "Smart rooms" as presented in Hewlett Packard's Cooltown project [KB2001]. The Smart Lecture Theatre aims to support users of lecture theatres namely lecturers and students in universities. The basic architecture of Smart Lecture Theatre system focuses on the ability for a user to fire queries from his/her device (such as mobile devices or desktop). Besides querying the Smart Lecture Theatre system, the users are able to perform tasks such as booking the lecture theatre and negotiating with other users to arrange swapping of bookings.

Each transaction corresponding to the lecture theatre is taken care of by an individual agent. Therefore, the Smart Lecture Theatre system uses multiple agents where each agent has a specific task assigned. Within the Smart Lecture Theatre architecture, the agents may either be stationary or mobile. Each agent will seek to perform and fulfil the task assigned. For example, if a student needs to find out the contact details of lecturer A, he/she will fire a query from his/her user device. Each student is represented by an unique user agent. Once the query has been triggered, the user agent creates query agent which will migrate to the "Smart Lecture Theatre" and attempt to get an answer to the student's query. Thus, it is obvious that the design of Smart Lecture Theatre requires mobile agents in order to move from the user device to the lecture theatre to accomplish the task-assigned. Further, if a lecturer A requires the lecture theatre for a specific time slot and the room has been booked by another lecturer B, the negotiation agent of Lecturer A will be required to travel to the user device of Lecturer B and request a possible swap of time slots.

This section documents the transition from the conceptual m-GAIA model of the Smart Lecture Theatre to the actual implementation using the Grasshopper Software Development Kit. This enables us to evaluate the transition from the conceptual m-GAIA methodology to an implementation and analyse how m-GAIA supports current mobile agent implementation environments/toolkits.

3.1 Implementation of the Smart Lecture Theatre System in Grasshopper

The SLT System as implemented in Grasshopper chiefly consists of two components – the user device on which the Staff, Student and the Administrator agents are created and reside, and the system device on which the LTAgent resides and interacts with the SLT database. Depending upon the type of user and the user needs, the QueryAgent, BookingAgent, NegotiationAgent or the UpdateAgent are initiated in the user device. These agents are mobile and migrate to the system device and interact with the LTAgent to process the queries input by the users. The students have the least privileges and can only query the SLT system for simple details such as unit details or lecturer details. The Administrator does the database administration and is able to perform updates on the database. The staffs have the most important functionality and are able to make lecture hall bookings by deploying mobile agents. If another lecturer has already made the booking, the lecturer requesting the booking can negotiate the booking by deploying the NegotiationAgent. While the system device is always up and running, and is registered with a *region registry,* the users can start up their devices as and when they want to use the system. Once they have finished using the system, they can choose to close the application. However in the case of the staff users, the agency

on the user device has to be up and running throughout the lifecycle of the system in order to facilitate negotiation of bookings. The SLT System setup in Grasshopper is indicated in *Figure 7* shown below.

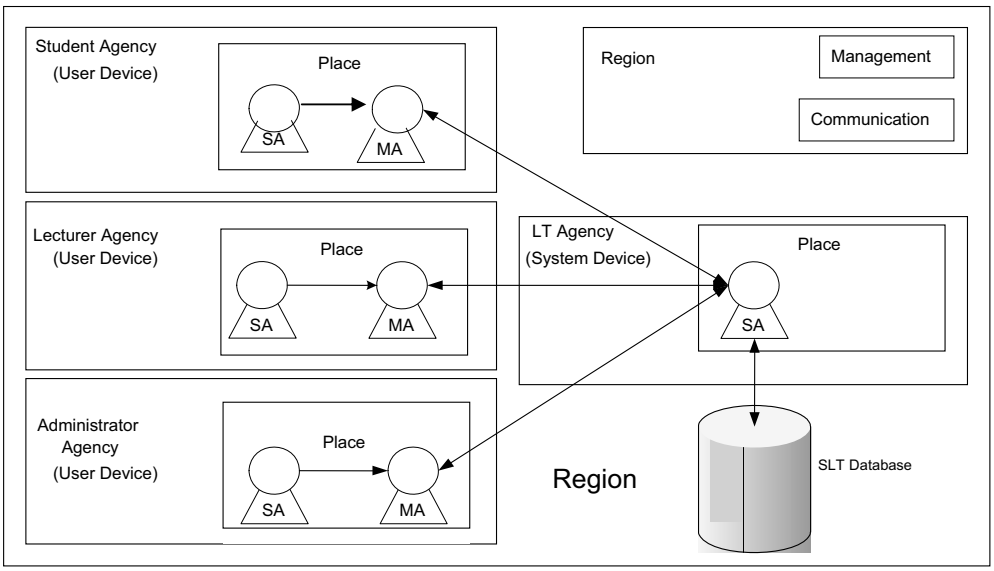

Figure 7: Component Structure of the SLT Structure in Grasshopper

3.2. Mapping m-GAIA to Grasshopper

The analysis phase of m-GAIA results in the *roles model* and the *interaction model*. From these abstract models, the design phase of m-GAIA provides us with *agent model, acquaintance model, services model* and *mobility model.* Starting from the *agent model*, there is a one-to-one correspondence between the agent types identified in m-GAIA and the actual Grasshopper agents that will be realised in the SLT system. Also there is a complete mapping of the mobility model into the mobility of the agents in the Grasshopper environment. In Grasshopper, the mobile agents are derived from the class de.ikv.grasshopper.agent.MobileAgent while the stationary agents are derived from the class de.ikv.grasshopper.agent.StationaryAgent. The services, which are derived from the *services model,* are directly implemented as methods in the grasshopper agents. The safety properties of the m-GAIA roles model are taken into consideration while defining these methods. Some of the methods will be executed during run-time only if the safety conditions hold true. The chief safety conditions that need to be met are the establishment of a connection between the different user devices and also the connection establishment with the SLT database. Whenever the connection fails to be established, an appropriate error message is displayed. While some of the services map onto methods within the agents, some of them are decomposed into a number of methods.

The activities translate into simple methods in the grasshopper agents. However the protocols, which represent the interaction between the different agents, are implemented in the Grasshopper platform by using the Communication Service that provides for interaction between the different agents by method invocation. In order to use the communication service the following steps have to be performed:

- Implementation of the server side (in this case, LectureTheatreAgent)
- Generation of the server proxy

- Implementation of the client side (the QueryAgent, the BookingAgent and the UpdateAgent)

The place types in the *mobility model* translate into places within the agencies in Grasshopper. The place in which the LectureTheatreAgent resides becomes the LT place, and the places in which the other agents reside become the UD places. The travel schema that is defined in the mobility model is realised in the SLT system during run time. The LTAgent assumes the *system role* as defined in the *role* model of the analysis phase. The QueryAgent, the BookingAgent, UpdateAgent and the NegotiationAgent assume the *interface* role and the Student, Staff and the Administrator agents take on the *user* role.

4. Conclusions and Future Directions

Mobile agents are increasingly being seen and used as a suitable technology to support distributed computing applications. Current agent-oriented software engineering methodologies do not support the explicit modelling of mobile agent systems. In this paper, we have presented our extension of the GAIA methodology to support modelling mobile agent systems and demonstrated its mapping to an implementation using the Grasshopper toolkit. This is but a first step towards building methodologies that support the analysis and design of mobile agent applications. There remain several open issues such as whether existing methodologies should be extended or new methodologies developed, determining the specific constructs for modelling mobility of agents (e.g. location, and itineraries).

5. References

[1] BAUMER C., BREUGST M., CHOY S., MABEDANZ T., 'Grasshopper – A Universal Agent Platform based on OMG MASIF and FIPA Standards' in URL:
http://www.cordis.lu/infowin/acts/analysys/products/thematic/agents/ch4.htm

[2] JUAN T., PEARCE A., STERLING L., 'ROADMAP: Extending the Gaia Methodology for Complex Open Systems', *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part I,* July 2002.

[3] KLUSCH M., ZAMBONELLI F., (ed) 'Cooperative Information Agents – Best Papers of CIA 2001', *International Journal of Co-operative Information Systems,* 2002 Vol. 11, No. 3 and 4.

[4] LING, S and LOKE, S.W.: Verification of Itineraries for Mobile Agent Enabled Interorganizational Workflow. Proc. Of the 4th Int. Workshop on Mobility in Databases and Distributed Systems. 2001. IEEE Computer Society. (ISBN 0-7695-1230-5). pp. 582-586.

[5] P. MORAITIS, E.PETRAKI, N.I. SPANOUDAKIS, 'Engineering JADE Agents with the Gaia Methodology', *Agent Technology Workshop 2002,* LNAI 2592, pp. 77-91, 2003, Springer-Verlag, Berlin, Heidelberg 2003.

[6] A. SELF, S.A.DELOACH, 'Designing and Specifying Mobility within the Multiagent Systems Engineering Methodology', *Special Track on Agents, Interactions, Mobility, and Systems (AIMS) at The 18th ACM Symposium on Applied Computing (SAC 2003),* March 9-12, 2003, Melbourne, Florida, USA.

[7] WEIβ, G. 'Agent Orientation in Software Engineering', *Knowledge Engineering Review,* 2002, Vol. 16, No.4, pp. 349-373

[8] WOOLDRIDGE, M., JENNINGS N.R., and KINNY, D., 'The Gaia Methodology for Agent-Oriented Analysis and Design', 2000, *Journal of Autonomous Agents and Multi-Agent Systems,* Vol.3, No. 3, pp. 285-312