

# Cross-Platform Generative Agent Migration

David R.A. de Groot, Frances M.T. Brazier and Benno J. Overeinder  
Intelligent Interactive Distributed Systems Group, Computer Science, Faculty of Sciences  
Vrije Universiteit Amsterdam, de Boelelaan 1081a, 1081 HV, Amsterdam  
Email: {DRA.de.Groot, FMT.Brazier, BJ.Overeinder}@few.vu.nl

**ABSTRACT:** In current agent systems agent migration is only possible between homogeneous systems supporting identical agent platforms, limiting an agent's possibilities considerably. This paper revisits the notion of generative migration. Generative migration entails migration of an agent blueprint, instead of complete code. This approach relies on homogeneity of libraries on different platforms to re-incarnate agents, but does not require homogeneity of platforms. Agent Factories are used to assemble agents at their destination, adapting an agent to its environment. This paper continues earlier work on generative migration by extending available theory and providing a demonstration and implementation of generative cross-platform agent migration using Agent Factories.

**KEYWORDS:** agents, mobility, agent platform, generative migration

## INTRODUCTION

Mobile agents traverse the Internet, e.g. searching for information on many different sites. These sites differ significantly. Over the last decade numerous agent platforms have been developed, each with their own specialties and peculiarities. The AgentLink<sup>1</sup> website currently lists over a hundred available platforms. Examples are JADE [1], Fipa-OS [2][3], AgentScape Operating System [4][5], Zeus [6], and Grasshopper [7]. Connecting several agent platforms results into a heterogeneous environment, see also the Agentcities<sup>2</sup> initiative. Such an environment changes over time, as new platforms are added and other connections (temporally) fail. Currently migration is only possible between homogeneous systems supporting identical agent platforms, limiting an agent's possibilities considerably. Ideally agents would be able to adapt to destination agent platforms.

This article focuses on an approach to adaptation based on the concept of generative migration using an Agent Factory for automated assembly. *Generative migration* is presented in [8]. The key idea of *generative migration* is not to move agent code and data but to base migration on an implementation-independent description of the agent. An *Agent Factory* [8][9][10] is a service capable of (re-) designing blueprints, finding appropriate building blocks and assembling these into a working agent: design, retrieval and assembly. An Agent Factory can be used to (a) (re-) design (i.e., creates a new blueprint) and adapt an agent (e.g. such that it is the agent can execute in another agent platform), and (b) to assemble and regenerate an agent at its destination adapted to its new environment. This paper discusses the need for generative migration using a description of an experiment in which generative migration has been implemented.

The structure of this document is as follows. Section 2 provides an introduction to the topic of agent migration and explains the principles of generative migration. Section 3 explains the experimental setup and introduces the agent platforms used. The fourth section presents related research and discusses alternative approaches to cross-platform agent migration. Section 5 ends this paper with a discussion and recommendations for future research.

## AGENT MIGRATION

Agent migration entails moving agents from one location to another. Two major types of agent migration can be identified: strong and weak migration. In strong migration an agent process and its execution context (execution state, program counter, etc.) are moved to the destination. In systems supporting strong migration, the migration is often completely transparent to the agent.

In weak migration two important aspects of the agent are distinguished: the agents' state and the agents' code. In the case of weak migration, agent state refers to the internal state and the private data of the agent. In weak migration, state and/or code can be migrated. Agent state migration entails saving the state of the agent in a format the destination can handle. The agent-code is made available on both sides so that an agent process can be created on the destination

---

<sup>1</sup> <http://www.agentlink.org>

<sup>2</sup> <http://www.agentcities.org>

platform, which is then initialized with the transferred agent state. This particular approach to weak migration is, for example, implemented in an extension of the Fipa-OS agent platform [11].

A number of different situations/scenarios can be distinguished with respect to agent migration as discussed in [8]. The *Homogeneous Migration Scenario* is the simplest form of migration when both the source and destination have the same interfaces and provide a similar environment for the agents on both sides. The interfaces are the same because source and destination locations run the same platform. To migrate agents, no changes to the agents' executable code need to be made. Homogeneous migration can be either strong or weak migration, but very few agent platforms support strong migration (for which all sites must also have exactly the same operating system, machine architectures, (version of) the programming language). Most of the currently popular agent platforms are Java-based and support weak mobility.

For the next scenario, the notion of agent platform instance needs to be introduced. Analogous to the programming-language instance concept, where more instances of a class can be used, multiple instances of a specific agent platform can exist in parallel. For example, the JADE agent platform can be started on two different servers. Each of those instances forms its own agent platform instance, thereby introducing the following scenario.

In the *Cross-Platform Migration Scenario* an agent migrates between instances of agent platforms. Two forms of cross-platform migration can be identified. First, homogeneous cross-platform migration between instances of the same type of platform: e.g. from Fipa-OS to Fipa-OS. In this case strong migration may still be possible; however, the authors are unaware of any implementations. Second, heterogeneous cross-platform migration between instances of different types of platform: e.g. from Fipa-OS to JADE. It is assumed that both platforms are programmed in the same programming language and therefore an agent's executable code does not need to change, because the destination's (virtual) machine is compatible. However, the interfaces (APIs) and machine architectures may differ, which complicates strong and weak migration. Different solutions to the cross-platform migration problem are possible [12]. Standardization is one solution, although in current practice this has failed because of interface incompatibility<sup>3</sup>.

Another solution entails that the destination platform has different agent servers [4]. Each agent server supports a different interface (i.e. it emulates a different platform) and can host agents that require this interface. A migrating agent can migrate to a platform if there is a suitable agent server available on the destination.

In addition to the above-mentioned scenarios a *Heterogeneous Migration Scenario* can also be identified in which agents move between platforms written in different programming languages. The interfaces differ almost by definition. Neither strong nor weak migration with Java-serialization is applicable since an agents' code is incompatible with the environment to which it migrates to, e.g. an Ajanta [13] agent (programmed in Java) moves to a DESIRE execution environment (programmed in Prolog) [8]. The agent's original source code is useless for generating executable code, because the compiled code is not supported at the destination. This is the most difficult situation for migration. A possible solution is code-translation [12]. However, not only the code needs to be translated, the agents may need to be adapted to a different interface as well. Research was done to automatically convert Java-based agents from Aglets to Voyager [14]. However, even with Java-based agents, this approach is not easy to implement and is only possible when the source code of the agents is available. Furthermore, it requires the development of ( $n^2 - n$ ) converters between  $n$  different platforms.

Another option for heterogeneous migration is *generative migration*. Generative migration can overcome the heterogeneous nature of the environment and has additional benefits for security and trust [8]. Our approach to generative agent migration makes use of an Agent Factory and is explained in the following section. Originally the Agent Factory was intended to be a (re-) design centre for agents; according to specific requirements it automatically engineers a new agent [8][9][10]. Another possible use is to employ the Agent Factory in a migration scenario, where the Agent Factory is used to generate a new 'incarnation' of an agent. This paper describes an implementation of this concept.

## PRINCIPLES OF GENERATIVE MIGRATION

The process of generative migration [8] is illustrated in Figure 1. It involves using a configuration of building blocks to describe the compositional structure of an agent. Such a description is called a *blueprint*. The blueprint describes an agent's structure and functionality. An agent also has a state (including private data) that is needed for its execution. Both the blueprint and state are described in a format independent of the operating system and agent system, e.g.

---

<sup>3</sup> For example, both the JADE and Fipa-OS comply to the FIPA (<http://www.fipa.org>) standards. An agent in Fipa-OS [2][3] calls the method `forward(ACL aclMsg)` to send a message, and an agent in JADE [1] calls the method `send(ACLMessage msg)`. If an agent written for Fipa-OS is started on JADE, its call to the method `forward()` results in an error. In this particular case, migration fails because the interfaces (APIs) are incompatible despite the compliance to a common standard (FIPA).

according to the XML syntax. An example of how a blueprint could be structured in XML is given in Figure 2. The blueprint and state descriptions are sent to the destination where an Agent Factory processes them. The Agent Factory re-builds the agent given the blueprint. Appropriate building blocks must be available at the destination platform, to enable regeneration of the agent's code. The building blocks are retrieved from a local repository (a database for reusable building blocks) and an agent is assembled. Finally, the regenerated agent is initiated with its state. Then, the agent is launched into the platform where it continues its execution.

As can be found in earlier work, see [8][9][10], using an Agent Factory for generative migration is based on a number of assumptions. The first assumption is that agents have a compositional structure. A compositional structure facilitates the possibilities of adding, removing and changing building blocks of an agent. Secondly, components and the compositional structure should be described in a (semi-) formal format, independent of an agent's implementation. The third assumption is that one or more libraries of re-usable components are available. The components of an agent are called *building blocks*. Building blocks are re-usable and simple to instantiate for use in newly (automatically) generated agents. Building blocks need to be stored and retrieved, in a local repository or a library.

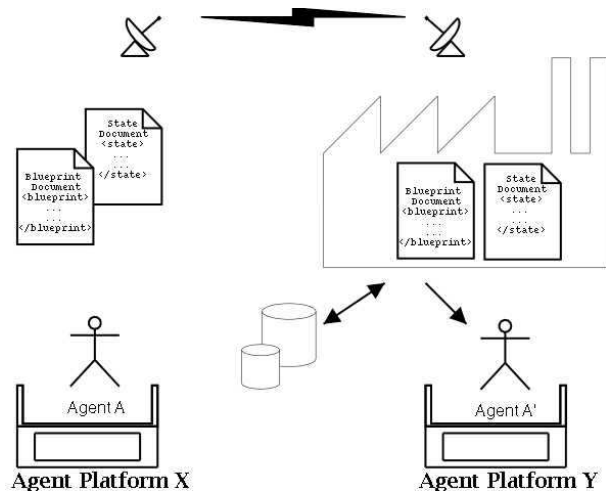


Figure 1: Principles of Generative Migration

```

<Blueprint>
  <BBconfigurationSet ID>
    <ConfigurationElements>
      <BuildingBlock>
        <BBid/>
        ... ..
      </BuildingBlock>
    </ConfigurationElements>
    <ConfigurationConnections>
      <OpenSlotFiller/>
    </ConfigurationConnections>
  </BBconfigurationSet ID>
</Blueprint>

```

Figure 2: Example XML structured blueprint

To model agents (independent of the implementation and in an automated process) and to reason about the models distinctions in the levels of detail in the design process are made. This requirement introduces a fourth assumption: conceptual and operational level descriptions are distinguished. Blueprints and building blocks exist both on conceptual and operational levels. A conceptual blueprint contains a (partial) conceptual specification defining a conceptual model of the agent. It is an abstract description of the structure, functionality and behaviour, which could be extended using UML [15]. Such a conceptual description may have one or more operational descriptions. A configuration of operational building blocks together defines a realization (implementation) of a conceptual model. Separation of the conceptual and operational levels allows more flexible (re-)design and implementation. In generative migration of agents the conceptual blueprint is sent to the destination and remains the same. The operational blueprint is re-assembled in the new environment.

Although the principles for generative migration can be used for other purposes than agent migration (e.g. software migration and update propagation), this paper focuses on the use of generative migration for moving agents.

## THE EXPERIMENT

The experimental setting consists of three agent platforms: AgentScape Operating System (prototype version), JADE (version 3.01b) and Fipa-OS (version 2.10). The goal of our experiment is to couple the platforms to enable the exchange of messages and the migration of agents following the principles on generative migration (i.e. a blueprint and state are transferred).

## AGENT PLATFORMS

Agent platforms provide an environment for the execution of agents. Often, an agent platform offers facilities and services to agents on the platform, for example life-cycle (starting, suspending, resuming, deleting, agents) and communication facilities and Directory Services (White and Yellow Pages). In addition, agent platforms can offer support for migration and security. Security is an intriguing and complicated issue in the world of agents and agent platforms, however it is not an issue of consideration in this paper.

The AgentScape Operation System (AOS) [4][5] includes an agent platform, a number of services (Generative Migration Facility, Directory Services), and support for application developers. The major challenge in the AgentScape project is to realize a scalable, secure, and fault tolerant system, that supports multiple distributed applications, heterogeneity (of agents, languages and operating systems), and multiple qualities of service. The AOS provides a platform in which mobile autonomous processes, the agents, can be managed.

JADE [1] stands for Java Agent Development Environment and is a FIPA-compliant agent platform. The main objective of JADE is to simplify the development of agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. As a secondary objective the JADE agent platform tries to optimize the performance of a distributed agent system implemented in the Java language. JADE consists of two parts: The first part is the runtime environment for FIPA-compliant multi-agent systems. The second is a framework for developing FIPA-compliant agent-systems. JADE supports weak agent migration by means of Java-serialization.

Fipa-OS [2][3] is an open source agent platform implemented in the Java programming language. The main purpose of the Fipa-OS agent platform is to provide a reference implementation for the standard specifications of the FIPA organization. A key focus of the platform is that it supports openness through a loose coupling between the platform components and compliance to the FIPA Agent Management reference model. Fipa-OS is being deployed in several application domains including virtual private network provisioning, distributed meeting scheduling and a virtual home environment. Note that the default releases of Fipa-OS do not support agent migration.

## ASSUMPTIONS

Given the Assumptions for Generative Migration, extra assumptions are necessary for the realization of Cross-Platform Generative Migration implementation:

- 1) Appropriate communication facilities in the agent platforms exist: e.g. facilities for sending ACL messages via HTTP or IIOP. A bidirectional channel for communication is needed for sending ACL messages (or other objects). The communication channel is used for requesting migration and sending/receiving information about the agents (e.g. blueprint and state documents);
- 2) There is a service for agent creation or regulated control over the agent creation process. By means of an agent creation process a new agent can be created on a target platform. Control over the creation process allows for the implementation of a service that starts and initializes a new agent.

In addition, for this prototype a choice was made with respect to language. The agent platform is Java-based and/or supports the execution of Java-based agents. A constraint on the programming language and/or operating systems is not directly necessary, but is useful to limit the domain of the research. It was not necessary to implement a new set of building blocks for languages/platforms involved. The agent platforms AOS, JADE and Fipa-OS satisfy the third condition: they all support Java agents. The test-results of the first two assumptions are presented in Table 1.

		<b>Communication Protocols</b>	<b>Creation Process</b>
<b>JADE</b>		HTTP, CORBA ORB (IIOP)	simple, can be isolated
<b>Fipa-OS</b>		HTTP, CORBA ORB (IIOP)	currently part of the GUI, difficult to isolate
<b>AOS</b>	prototype	(internal only)	Service in the platform
	1.0	HTTP	Service in the platform

**Table 1 Test results**

The test results show that all platforms support communication based on the HTTP protocol. Exchanging messages was, however, not trivial<sup>4</sup>.

Agent creation was also more difficult than expected. The agent creation process is difficult to isolate in Fipa-OS. That is because it is integrated in the User Interface classes and severely coupled with other platform parts. Implementation of a service for automatic agent creation and initialization does not seem to be possible in the current version of Fipa-OS. However, the creation process is available in JADE and AOS.

Combining these results, for the choice was made to illustrate cross-platform generative migration for migration from Fipa-OS to the JADE agent platform.

## SCENARIO

The scenario focuses on the technical aspects of generative agent migration. It describes the parties involved and the interactions between the parties. The environmental setting consists of two Agent Platforms: Fipa-OS and JADE. A number of services is assumed to be present on both platforms: a Directory Facilitator (DF) and a Generative Migration Service (GMS). The Directory Facilitator (DF) is a service standard present in the agent platforms. The GMS is our own implementation and its primary task is to take care of migration and agent creation and initialization. The GMSs are registered at the local DF and can be found by agents and other services on both platforms if the DFs are cross-registered. Furthermore an agent is present that requests to be migrated.

Thus, an agent sends a migration request to the local GMS for migration to a remote platform. The GMS contacts the remote GMS and forwards the request. An agent receives an acknowledgement of the migration request, or denial with an indication of the reason for denial (e.g. no GMS on target location, building-blocks not present on target or an agent with (requested) identical name exists). On acknowledgement an agent hands over its blueprint and state to the local GMS. The local GMS contacts the remote GMS for creation of a new agent with a given blueprint, state and agent-name. The remote GMS needs to be able to assemble a new agent based on the blueprint of the requesting agent, it must have capabilities and permission to create a new agent on the platform and the means to initialize the new agent with the state of the original agent. Once a new agent is running the agent in the source platform can be stopped and removed.

## THE IMPLEMENTATION

As explained in Section 2, a compositional agent can be described in a blueprint document, written in an independent format that can be exchanged between locations. Additionally, state information can be used to initialize a newly generated agent on the target location.

To implement generative agent migration in agent platforms, two frameworks had been developed: a conceptual framework and an operational framework. The conceptual framework is based on the DESIRE modelling framework [16] and has three kinds of structure elements: components, links, and information types. Each component has an input and an output buffer and components define their input and output information types. Links can be placed between components to connect input and output buffers. A link transfers information elements (instances of information types) from an output buffer to an input buffer. Which information types a link transports can be predefined. In this framework only component structure elements can contain other structure elements.

On the operational level, an implementation-language specific framework is needed. For building-block components implemented in the Java programming language a framework had been developed which has a one-to-one mapping with the conceptual framework. The operational framework is used to dynamically load Java components via the provided mechanisms for class loading.

A compositional agent is embedded in a wrapper, the wrapper is an extension of the base-agent of an agent platform, e.g. in JADE it is `jade.core.Agent` and in Fipa-OS `fipaos.agent.FIPAOSAgent`.

## EVALUATION

The current implementation demonstrates heterogeneous cross-platform agent migration from Fipa-OS to JADE. In contrast, with the same approach, homogeneous cross-platform migration is possible, e.g. migration between instances of the same platform. Generative agent migration from JADE-to-JADE platforms has been tested successfully.

---

<sup>4</sup> The Fipa-OS platform needs special configuration settings to connect and communicate with a remote platform. The standard setup procedure uses a handshake-protocol and expects the other side to be a Fipa-OS platform. Since JADE isn't programmed to react to the Fipa-OS handshake, the setup fails. One solution to make the correct settings is to connect two Fipa-OS platforms first and then shut down one of them, replacing it with JADE. Another solution is to manually edit the `acc.profile` file, which is located in the profiles directory.

## RELATED RESEARCH

Related research with respect to this paper concerns the agent factory and cross-platform agent migration initiatives. A short summary of related approaches is given, followed by a comparison.

In this paper an Agent Factory is a service to design, adapt and (re-) assemble agents from building blocks. The Agent Factory to which this paper refers is the Agent Factory described in [8][9][10]: the IIDS Agent Factory. There are other agent factories. The Factory of the Agents [17][18] currently being developed to provide "*a cohesive framework that supports a structured approach to the development and deployment of agent oriented-applications*", providing extensive support for the creation of Belief-Desire-Intention (BDI) agents.

The Agent Factory described in [19][20] is being designed to be a service capable of providing designers with a tool for building agents, and a repository of patterns that can be used to add new capabilities. Agents in this project are developed according to the PASSI design methodology [20], with which models of multi-agent systems and agent interactions are specified and expressed in UML.

The IIDS Agent Factory has been designed for automated design and redesign of single agents, whereas the two other agent factories are being designed to assist human designers designing multi-agent systems. However, some similarities can be found in the approaches to agent design and the assembly process.

Also related to this research are approaches to interoperability and cross-platform migration. Known implementations are based on wrappers, intermediate interface layers, and agent servers. Especially worth mentioning are the Guest [12][21] and Monads [22] research efforts.

The developers of Guest [12][21] propose a middleware-based model that introduces an intermediate layer for the support of their agents on top of an existing agent platform. Regardless of the underlying platform, if it supports the Guest Layer, Guest Agents can be executed. The goals of the project are 1) to allow interoperability between platforms, i.e. allowing agents to run, communicate and move between them. 2) To provide a uniform view of those platforms, i.e. agents can be written and manipulated without considering the kind of servers they will run on. 3) To add new adaptive features to the platforms, i.e. plug-in mechanisms for dynamic extensions. Though still an experimental prototype, Guest enables interoperability between Voyager, Aglets, Grasshopper and JADE. However, no source-code or libraries are available for public or research usage.

The Monads project [22] concentrates on the needs of nomadic users and adaptability. By adaptability they primarily mean the ways in which services adapt themselves to properties of terminal equipment and to characteristics of communications. This involves both mobile and intelligent agents as well as learning and predicting temporary changes in the available Quality-of-Service along the communications paths. The goal of Monads is to design an efficient and reliable software architecture based on adaptive services and agents, and to develop prototypes based on that architecture. The agents themselves are not adaptive, only used to steer changes in e.g. quality of service.

The basic approach to mobile agents in Monads is a separation of an agent into a head and a body. The body handles the agent-programming interface of each agent platform, and a head can be placed on top of it. The agent-head can migrate via a service called the Monads Agent Gateway (MAG). The Monads approach has been implemented on JADE, Voyager and Grasshopper agent platforms. Unfortunately, no source-code or libraries are available for public or research usage.

There are some differences and similarities with our approach. The Guest Interface Layer defines a kind of generic agent interface that can be supported on other platforms. This differs with the approach explained in this paper because our agents are adapted (i.e. assembled) before execution on another platform takes place and agents in Guest are not adapted. A similarity with Monads and our approach concerns one of the goals of Guest: to provide a uniform view of the underlying platform.

In Monads an agent consists of two parts: a head and a body. In our approach a similar division into agent-head and agent-body can be made. In contrast with the Monads agent-head our agent consists of multiple components and can be migrated using generative migration whereas Monads does not make this distinction and uses Java object serialization for migration. Additionally the functionality of their MAG-service is comparable with our Generative Migration Service.

## DISCUSSION

Earlier research showed the possibility of homogeneous generative migration in the AgentScape OS (AOS) agent platform [4][5]. This paper presents a demonstration of heterogeneous inter-platform migration (cross-platform migration between different types of agent platforms) using generative migration.

Several alternative design choices could have been made

- 1) Agents could take care of the state transfer themselves if the original agent is reactivated after a copy of the agent has been transferred.

- 2) The GMS can be a centralized service that is registered at both platforms, or be present in both platforms. In the above scenario one GMS would be sufficient if only migration to the target platform is required. Either a delete-request for the migrating agent would be needed for the management service of the source platform, or the agent would need to request the delete.
- 3) In this prototype a simple directory service sufficed. Cross-registration of Directory Facilities (the FIPA Directory Service), however, may be preferable, but requires uniformity of directory service standards.

Future research will address these aspects and will also include generative migration between AgentScape OS and JADE. Additionally, the next challenge with respect to generative migration is a demonstration of a heterogeneous migration scenario with building blocks in different programming languages.

## ACKNOWLEDGEMENTS

This paper is based on David de Groot's Master's thesis. The authors thank the graduate students Hidde Boonstra, David Mobach, Oscar Scholten and Mark van Assem for their explorative work, and Niek Wijngaards for his contribution to this paper. The authors are also grateful to the support provided by Stichting NLnet<sup>5</sup>.

## REFERENCES

- [1] Bellifemine, F. (CSELT S.p.A.), Poggi, A. (DII – University of Parma), Rimassa, G (DII – University of Parma), “Developing multi agent systems with a FIPA-compliant agent framework”. In: *Software - Practice And Experience*, 2001 no. 31, pagg 103-128.
- [2] Poslad S., Buckle, P. and Hadingham R. (2000) “The FIPA-OS Agent Platform: Open Source for Open Standards”. In: *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, UK, pages 355-368, 2000.
- [3] Poslad S., Buckle, P. and Hadingham R. (2001) “Open Source Standards and Scalable Agencies”. Presented at the *Autonomous Agents 2000 Workshop on Infrastructure for Scalable Multi-Agent Systems*, Spain, June 2000. Also printed in *Lecture Notes in Computer Science*, Springer-Verlag Heidelberg, ISSN: 0302-9743, Volume 1887 / 2001, January 2001.
- [4] Overeinder, B. J., Posthumus, E. and Brazier, F.M.T. (2002) “Integrating Peer-to-Peer Networking and Computing in the AgentScape Framework”, In: *Proceedings of the 2nd IEEE International Conference on Peer-to-Peer Computing*, pp. 96-103.
- [5] Wijngaards, N.J.E., Overeinder, B.J., Steen, M. van and Brazier, F.M.T. (2002), Supporting Internet-Scale Multi-Agent Systems, In: *Data and Knowledge Engineering*, Vol. 41, Number 2-3, pp. 229-245.
- [6] Collis, J.C., Lee, L.C., (1998) “Building Electronic Marketplaces with the ZEUS Agent Toolkit”, AMET'98, Workshop on Agent Mediated Electronic Trading, Minneapolis/St Paul, USA, May 10, 1998.
- [7] Bäumer, C., Breugst, M., Choy, M., and Magedanz, T., “Grasshopper - a universal agent platform based on OMG MASIF and FIPA standards”. In: Ahmed Karmouch and Roger Impley, editors, *First International Workshop on Mobile Agents for Telecommunication Applications (MATA '99)*, pages 1-18, Ottawa, Canada, October 1999. World Scientific Publishing Ltd.
- [8] Brazier, F.M.T., Overeinder, B.J., Steen, M. van, and Wijngaards, N.J.E. (2002) “Agent Factory: Generative Migration of Mobile Agents in Heterogeneous Environments” In: *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002)*, pp. 101-106.
- [9] Splunter, S. van, Wijngaards, N.J.E., Brazier, F.M.T., “Structuring Agents for Adaptation” In: *Adaptive Agents and Multi-Agent Systems*, (Alonso, E., Kudenko, D., Kazakov, D. Ed.), *Lecture Notes in Artificial Intelligence (LNAI) 2636*, Springer-Verlag Berlin. 2003.

---

<sup>5</sup> <http://www.nlnet.nl>

- [10] Brazier, F.M.T. and Wijngaards, N.J.E. (2001), Automated Servicing of Agents, In: *AISB Journal*, Vol. 1, Number 1, pp. 5-20 (Special Issue on Agent Technology).
- [11] Mäkeläinen, M. (2000) "Agent Mobility support for FIPA-OS". Project report Bachelor of Science in Computing Systems, Nottingham Trent University, Department of Computing.
- [12] Magnin, L., Pham, T. V., Dury, A., Besson, N. and Thieffaine, A. "Our Guest Agents are Welcome to Your Agent Platforms". In Proceedings of the ACM Symposium on Applied Computing (SAC 2002), pp. 107-114. Madrid, Spain, March 10-14, 2002.
- [13] Tripathi, A., Karnik, N., Vora, M., Ahmed, T., Singh, R. (1999) "Mobile Agent Programming in Ajanta", In Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS '99)
- [14] Tjung D., Tsukamoto, M. and Nishio S., (1999) "A converter Approach for Mobile Agent System Integration: A Case of Aglet to Voyager", proceedings of the First International Workshop on Mobile Agents for Telecommunication Applications (MATA 99), Ottawa, Canada, October 6-8, 1999, pp. 179-195.
- [15] Fowler, M., Scott, K., "UML Distilled", 2<sup>nd</sup> Edition, Addison Wesley Longman, 2000.
- [16] Brazier, F.M.T., Jonker, C.M. and Treur, J. (2002), Principles of Component-Based Design of Intelligent Agents, In: *Data and Knowledge Engineering*, Vol. 41, pp. 1-28
- [17] Collier, R.W., O'Hare, G. M. P., (1999) "Agent Factory: A Revised Agent Prototyping Environment", 10th AICS Conference, Irish Artificial Intelligence and Cognitive Science Conference, Cork, Ireland, 1999.
- [18] Collier, R.W., O'Hare G.M.P., Lowen, T., Rooney, C.F.B., (2003) "Beyond Prototyping in the Factory of the Agents", 3rd Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'03), Prague, Czech Republic, 2003.
- [19] Cossentino, M. (2002) "Different perspectives in designing multi-agent systems" - AGES '02 workshop at NODE02, 8-9 October 2002 - Erfurt, Germany.
- [20] Cossentino, M., Burrafato, P., Lombardo, S., Sabatucci, "Introducing Pattern Reuse in the Design of Multi-Agent Systems", AITA'02 workshop at NODE02 - 8-9 October 2002 - Erfurt, Germany.
- [21] Magnin, L., Snoussi, H., Pham, V. T., Dury, A. and J.-Y. Nie. "Agents Need to Become Welcome" In Proceedings of the 3rd International Symposium on Multi-Agent Systems, Large Complex Systems, and E-Businesses (MALCEB'2002). Erfurt/Thuringia, Germany, October 8-10, 2002.
- [22] Misikangas, P. and Raatikainen, K., (2000) "Agent Migration Between Incompatible Platforms". In the 20th International Conference on Distributed Computing Systems (ICDCS 2000), 10-13 April 2000, Taipei, Taiwan, Republic of China.