

Cross-Platform Generative Agent Migration

An Agent Factory Approach

Abstract

In some agent applications agents need to move between locations to perform their tasks. Agent migration, however, is often complicated by the heterogeneous nature of the agent environment. For example, the platform from which an agent migrates (source-platform) may not be compatible with the platform to which the agent migrates (destination platform). Different solutions for *cross-platform agent migration* are possible. This thesis addresses one such solution, called *generative migration*. Instead of transporting the agent itself, a blueprint of the agent is sent to its destination. There, an Agent Factory regenerates the agent using its blueprint. This thesis continues earlier work on generative migration by extending available theory and providing a demonstration and implementation of generative cross-platform agent migration.

Master's Thesis

David R.A. de Groot

Student number: 1044273

E-mail: davidra@cs.vu.nl

January, 2004

Intelligent Interactive Distributed Systems

Computer Systems Section

Department of Computer Science

Faculty of Sciences

Vrije Universiteit Amsterdam

de Boelelaan 1081a

1081 HV Amsterdam

The Netherlands

Supervisor:

Prof. Dr. F.M.T. Brazier

Second reader:

Dr. B.J. Overeinder

Advisors:

Dr. N. J. E. Wijngaards

Drs. S. van Splunter

Nederlandse samenvatting

In applicaties waarin agenten een rol spelen is het vaak gewenst dat agenten zich kunnen verplaatsen. Helaas wordt agent migratie vaak beperkt door de heterogeniteit van de omgeving. Zo kan bijvoorbeeld het platform waar de agent naartoe wil migreren (het doel platform) een ander type platform zijn en geen goede aansluiting bieden voor de agent (interface incompatibiliteit). Verschillende oplossingen voor *cross-platform migratie* van agenten zijn mogelijk. Deze scriptie gaat over één zo'n oplossing: *generatieve migratie*. Daarbij wordt niet de agent zelf verplaatst maar wordt een bouwtekening (blueprint) overgestuurd. Vervolgens wordt op het doel platform de agent opnieuw gebouwd door een zogenaamde Agent Factory.

Dit onderzoek is een vervolg op eerder onderzoek naar generatieve migratie en geeft een uitbreiding op de theorie. Daarnaast is een demonstratie geïmplementeerd waarmee de mogelijkheid van cross-platform generatieve migratie wordt aangetoond.

Table of Contents

Chapter 1 Introduction.....	5
1.1 Context of the research.....	5
1.2 Research Focus.....	5
1.3 Research Questions.....	6
1.4 Overview.....	7
Chapter 2 Agent Migration.....	8
2.1 Agents.....	8
2.2 Agent Migration.....	9
2.2.1 Migration scenarios.....	9
Homogeneous Migration Scenario.....	9
Cross-Platform Migration Scenario.....	10
Heterogeneous Migration Scenario.....	11
2.3 Generative Migration.....	11
Agent-Regeneration Migration Scenario.....	12
2.3.1 Principles of Generative Migration.....	12
2.3.2 Frameworks.....	14
Chapter 3 Interoperability of Agent Platforms.....	15
3.1 Agent Platforms.....	15
3.1.1 AOS.....	15
3.1.2 JADE.....	16
The JADE Agent Model.....	17
JADE Platform organisation.....	18
3.1.3 Fipa-OS.....	19
3.1.4 Agent Platform Topologies.....	20
3.2 Interoperability.....	21
FIPA.....	22
OMG MASIF.....	23
3.3 Cross-Platform Generative Migration.....	24
Chapter 4 Practical Work.....	26
4.1 Design.....	26
4.1.1 Scenario's.....	26
4.1.2 Interaction Diagram.....	28
4.1.3 Class diagram for IAOF integration.....	29
4.2 Implementation Problems.....	31
4.2.1 Translating information.....	31
4.2.2 The AgentImage.....	31
4.2.3 Package Integration.....	32
Chapter 5 Related Research.....	33
5.1 Agent Factories.....	33
5.1.1 Agentfactory.com.....	33
5.1.2 The Italian Agent Factory.....	34
5.1.3 The German Agent Factory.....	34
5.1.4 Comparison.....	34
5.2 Agent Modelling.....	34
5.3 Alternative Approaches for Interoperability.....	36
5.3.1 Guest Agents.....	36

5.3.2 Monads Project.....	36
5.3.3 SeMoA.....	37
5.3.4 GMAS.....	38
5.3.5 Comparison.....	39
Guest Agents.....	39
Monads.....	39
SeMoA.....	39
GMAS.....	39
Chapter 6 Discussion.....	41
6.1 Overview.....	41
6.2 Future Work.....	42
6.2.1 Security and trust.....	42
6.2.2 Agent Identity.....	42
6.2.3 Messaging.....	42
6.2.4 Fault-tolerance.....	42
6.2.5 Homogeneous cross-platform migration.....	43
6.2.6 Self-aware agents.....	43
6.2.7 Heterogeneous migration.....	43
References.....	44
Appendix I.....	49

Trinity: It's the question that drives us, Neo. It's the question that brought you here.
You know the question just as I did.
Neo: What is the Matrix?

[The Matrix]

Chapter 1 Introduction

In this chapter the reader is introduced to the research of this MSc Thesis. First, the context of the research is sketched, followed by a more specific definition of the research focus and goals. This chapter ends with an overview of the thesis.

1.1 Context of the research

This research is performed in the context of IIDS¹ research at the Vrije Universiteit, Faculty of Exact Sciences. IIDS stands for Intelligent Interactive Distributed Systems and professor Dr. Frances Brazier heads the group. Originally part of the Artificial Intelligence Research group, it recently became part of the Computer Systems group. The NLnet Foundation actively pursued the formation of IIDS, supports the research and has since the start supported the group in many ways.

IIDS' research programme focuses on support for the development of heterogeneous, secure, flexible and adaptable architectures for intelligent interactive distributed systems (for example large agent systems). The research programme includes the AGENTSCAPE-project, which focuses on the design and development of middleware to support large-scale agent systems, including services such as an Agent Factory and agent Directory Services. The AOS (AgentScape Operating System) provides a distributed environment for agents. Such environments for agents are often referred to as *agent platforms*, and provide agents with basic facilities such as communication and locating other agents (Directory Services). Additional to these basic facilities agent platforms can offer facilities for migration and security.

1.2 Research Focus

Several methods for agent migration have been developed over the past decade, a new approach, called *generative migration*, has recently been presented by the IIDS group. See the articles (Brazier, Overeinder, Steen and Wijngaards, 2002a; Brazier, Overeinder, Steen and Wijngaards, 2002b). Additionally, Van Assem performed a MSc Thesis research on the subject of generative migration; see (Assem, 2003).

The key idea of *generative migration* is not to move an agent itself but a blueprint that describes an agent independent of its implementation. The blueprint describes an agent as a configuration of *building blocks* (Brazier and Wijngaards, 2001; Brazier and

¹ <http://www.iids.org>

Wijngaards, 2002). Appropriate building blocks must be available at the destination platform, to enable regeneration of the agent's code. Note that the subject of *security and trust* is not considered in this thesis.

An *Agent Factory* can be used for generative migration of agents. An Agent Factory is a service capable of (re-)designing blueprints, finding appropriate building blocks and assembling these into a working agent. The concept of the Agent Factory is introduced in (Brazier and Wijngaards, 2001). Additional information can be found in (Brazier and Wijngaards, 2002; Brazier, Overeinder, Steen and Wijngaards, 2002b; Splunter, Wijngaards, Brazier, 2003).

An Agent Factory supports generative agent migration in two ways: an Agent Factory (a) (re-) designs (i.e., creates a new blueprint) and assembles an agent such that it is possible to generatively migrate the agent, and (b) regenerates the agent at its destination. Agent Factories are not a main topic of this thesis. This thesis does not investigate how an Agent Factory's design process produces a blueprint as mentioned in (a). Agent Factories and blueprints for agents are assumed to be available.

A detailed description of the current version of the Agent Factory, its inner processes and strategies can be found in (Splunter, 2002). The foundation for defining and creating building blocks, as well as assembling agents (for this version of the Agent Factory) is described in (Scholten, 2003).

This thesis continues earlier research by exploring how these concepts can be used in the context of *cross-platform agent migration*, which is, moving an agent from one agent platform to another.

1.3 Research Questions

The research goal of the thesis is to demonstrate that generative migration of compositional agents is a possible solution for cross-platform agent migration. To approach the problem, the following questions need to be answered:

- What methods for migration of agents are currently used?
- What is *generative migration*? What are the pre-conditions (assumptions)? How is it achieved?
- What are *agent platforms*? What is *interoperability* of agent platforms?
- What approaches to cross-platform migration of agents exist and are used in implementations?
- What are the pre-conditions (assumptions) for those approaches?
- How can generative migration be used for cross-platform migration of agents?
- Is generative migration a good/practical/feasible alternative?

Finding answers to these questions is the focus of this research project. Apart from a theoretical exploration of the field, a practical implementation has been explored. The practical part involves a prototype implementation to show that cross-platform generative migration is possible. It is based on a re-usable design that can be implemented for other agent platforms as well.

1.4 Overview

The structure of this document is as follows. Chapter 2 gives an introduction to the topic of agent migration and explains the principles of generative migration. Chapter 3 discusses agent platforms and interoperability of agent platforms. Chapter 4 provides examples of application scenarios and covers the design and implementation of the prototype. The 5th Chapter presents related research and discusses alternative approaches to cross-platform agent migration. Chapter 6 ends this thesis with a discussion and recommendations for future research.

Chapter 2 Agent Migration

This chapter begins with a brief introduction to agents followed by a description of agent migration. A special type of agent migration, called *generative migration*, is discussed: the migration method that is central to this thesis.

2.1 Agents

Different communities including Artificial Intelligence (AI) and Computer Science (CS) have studied agents and agent systems for the past few decades. Research from the perspective of AI most often focuses on different aspects than CS-oriented research (Wijngaards, Overeinder, Van Steen and Brazier, 2002).

In Artificial Intelligence research, one of the most instructive and widely accepted descriptions of agenthood distinguishes two *notions of agency* (Jennings and Wooldridge, 1995). In the *weak notion of agency*, an agent is any hardware or software-based computer system that has the following properties:

1. *autonomy* (it has control over its own actions and state);
2. *social ability* (it can communicate with other agents);
3. *reactivity* (it reacts to changes in its environment);
4. *pro-activeness* (it makes plans to reach its goals and can take initiative to pursue these).

The *strong notion of agency* has more specific characteristics than the weak notion. Besides the properties of the weak notion, an agent should also be either conceptualised or implemented using *mentalistic* notions (knowledge, belief, intention, etcetera).

Unlike the properties of weak agency, mobility is an orthogonal property of agents. Or as Nwana puts it: “mobility is neither a necessary nor sufficient condition for agenthood” (Nwana, 1996). In some cases, agents can do without, but mobility has many potential benefits. These can be grouped into three categories: *performance*, *resource access* and *security* (Brazier, Overeinder, van Steen and Wijngaards, 2002).

In addition to these properties AI researchers are often interested in the internal models of the agents, in multi-agent systems and the application of AI techniques. AI-techniques can be applied inside the agents or by means of the agents (e.g. self-organizing systems).

From a Computer Science perspective the notions of agents as mentioned above are not very relevant. An agent is often considered to be an autonomous process, a piece of running code with data and state (Wijngaards, Overeinder, Van Steen and Brazier, 2002; Tanenbaum and Van Steen, 2002). However, the mobility property of agents is interesting to the computer science community. Agent migration is based on process and code migration, which stems from the field of Distributed Systems (Fuggetta, Picco and

Vigna, 1998). Agent migration means that the process or code that represents the agent is moved over a network from one computer to the other.

2.2 Agent Migration

Agent migration entails moving agents from one location to another. A location can possibly hide multiple computers, a typical issue originating from Distributed Systems. Two major types of agent migration can be identified: strong and weak migration. In strong migration the agent process and its process 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. NOMADS (Suri et al., 2000), ARA (Peine, 2002) and D'AGENTS (Gray, Cybenko, Kotz, Peterson and Rus, 2000) are agent systems that support strong migration. Despite the advantages of strong mobility, strong mobility requires to completely homogeneous environments. A *homogeneous environment* consists of locations that all have exactly the same operating system and agent system and the same version of the programming language on all locations.

In weak migration two important aspects of the agent are distinguished: the agents' state and the agents' code. In the case of weak migration, with *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 assumed to be available on both sides so that an agent process can be created on the destination, which is then initialized with the original state. This particular approach to weak migration is, for example, implemented in an extension of the FIPA-OS agent platform (Mäkeläinen, 2000).

Weak migration in the form of code migration implies that an agent process is started on the destination based on the programming code (compiled code and/or instantiated). After migration the agent always starts from a pre-specified method, e.g. in the JADE agent platform the `afterMove()` method of an agent is used for this purpose. In JADE a combination of code and state migration is implemented by means of the JAVA serialization technology. The JAVA serialization technology provides a mechanism to transport instances of objects. Since these instances are based on programming code this approach also falls into the category of weak migration.

2.2.1 Migration scenarios

Strong and weak migration are not always possible. Based on the possibilities for agent migration, a number of different scenarios are identified. These scenarios describe practical situations where agent migration could be applied and are adapted from (Brazier, Overeinder, Van Steen and Wijngaards, 2002a; (Brazier, Overeinder, Van Steen and Wijngaards, 2002b; Assem, 2003).

Homogeneous Migration Scenario

This is the simplest form of migration: 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 are part of the same platform. To migrate agents, no changes to the agents' executable code need to be made. If an agent has been written in an interpreted language, e.g.

JAVA, executable code can be run on a compatible *virtual machine* available at the destination. If an agent is written in a compiled language, e.g. C++, the executable code needs to be compatible with the destination's machine architecture and operating system.

In the homogeneous migration scenario is possible to support strong as well as weak migration. Very few agent platforms support strong mobility; examples are NOMADS (Suri, Bradshaw, Breedy, Groth, Hill, Jeffers, Mitrovich, Pouliot and Smith, 2000), ARA (Peine, 2002) and D'AGENTS (Gray, Cybenko, Kotz, Peterson and Rus, 2002). Most of the currently popular agent platforms are JAVA-based and support weak mobility only. For example: JADE (Bellifemine, Poggi and Rimassa, 2001), and GRASSHOPPER (Bäumer, Breugst and Choy, 1999).

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 location, thereby introducing the following scenario.

Cross-Platform Migration Scenario

In this situation, an agent migrates between instances of agent platforms. Each platform instance forms a location. It is possible to perform cross-platform migration homogeneously (instances of the same type of platform: e.g. FIPA-OS to FIPA-OS) or heterogeneously (instances of different types of platform: e.g. FIPA-OS to JADE). It is assumed that both platforms are programmed in the same programming language. Therefore an agent's executable code does not need to change, because the destination's (virtual) machine is compatible. However, in the case of heterogeneous cross-platform migration, the interfaces (APIs) may differ, which complicates the migration.

Different solutions to the migration problem are possible (Magnin et al, 2002a; Magnin et al, 2002b). Standardization is one solution, although in current practice this fails miserably. For example, both the JADE and FIPA-OS comply to the FIPA standards. An agent in FIPA-OS (Poslad, Buckle, and Hadingham, 2000; Poslad, Buckle, and Hadingham, 2001) calls the method `forward(ACL acLMsg)` to send a message, and an agent in JADE (Bellifemine, Poggi and Rimassa, 2001) calls the method `send(ACLMessage msg)`. If an agent written for FIPA-OS is started on JADE (if this was technically possible), its call to the method `forward()` would result in an error. In this particular case, migration fails because the interfaces (APIs) are incompatible despite the compliance to a common standard (FIPA).

A commonly applied solution to overcome interface differences is to provide an agent with a *wrapper*. The wrapper software hides the differences in the platform interface the agent expects and the interface the destination platform offers. The wrapper is placed between agent and platform, and 'translates' procedure calls of the agent to the platform to equivalent calls of the destination platform, and vice versa. See e.g., the Adapter / Wrapper pattern in (Gamma, Helm, Johnson and Vlissides, 1995). This approach is for example taken in GUEST AGENTS (Magnin et al, 2002a; Magnin et al, 2002b) and GMAS (Grimstrup, et al., 2002). However, they present their approach as intermediate interfacing layers between the agent platform and the agents.

A different solution entails that the destination platform has different *agent servers*.

Each agent server supports a different kind of platform 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 it has a suitable agent server (Overeinder, Posthumus and Brazier, 2002).

Another possible solution is to use *generative agent migration* with the help of an Agent Factory, which is the approach described in this thesis.

Actually, more specific terms should be used for the homogeneous and cross-platform migration scenarios; the term *intra-platform migration* is preferred instead of homogeneous migration. That is because the agent stays inside the same agent platform whereas the term homogeneous migration does not enforce this. Then, the two forms of cross-platform migration can be better characterized with *homogeneous* and *heterogeneous inter-platform migration*.

Heterogeneous Migration Scenario

In this case agents move between platforms written in different programming languages, additionally, interface incompatibilities are encountered. Therefore an agents' code is incompatible with the environment to which it wants to migrate to, e.g. an AJANTA agent (programmed in JAVA) moves to a DESIRE execution environment (Brazier, Overeinder, Van Steen and Wijngaards, 2002a). The agents' 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 (Magnin et al, 2002a). 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 (Tjung, Tsukamoto and Nishio, 1999). 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.

In the situation of heterogeneous migration, an alternative to code-conversion is generative migration. Note that generative migration is also applicable to the other scenarios.

2.3 Generative Migration

Originally the Agent Factory was intended to be a (re-) design centre for agents; according to specific requirements it automatically engineers a new agent (Brazier and Wijngaards, 2001). 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 (Brazier, Overeinder, Steen and Wijngaards, 2002a). This new approach is called *generative migration* and has been developed by the IIDS research group.

Although the principles for generative migration can be used for other purposes than agent migration (e.g. software migration and update propagation), we focus on the use of generative migration for moving agents. Based on the scenario for Heterogeneous Migration we can continue with a scenario for generative migration.

Agent-Regeneration Migration Scenario

In this situation, the agent migrates to a location where the agent cannot directly be executed. For example because the agent's executable code is not compatible with the destination's (virtual) machine and/or its interfaces (this is the case in heterogeneous and cross-platform migration scenarios). The solution is to regenerate the agent using a *blueprint* that describes the agent independent of its implementation (Brazier and Wijngaards, 2001; Brazier, Overeinder, van Steen and Wijngaards, 2002a and 2002b; van Splunter, Wijngaards and Brazier, 2003).

Apart from Van Assem's implementation (Assem, 2003), which is based on the prototype implementation (version 0.43) of the AGENTSCAPE OS (AOS), the author is not aware of any other platform supporting Agent-Regeneration Migration. Note that the term 'Agent-Regeneration Migration' is interchangeable with 'Generative Migration', however 'Agent-Regeneration Migration' is used in the source material for this particular scenario.

Our approach to generative agent migration is by means of an Agent Factory (AF) and is explained in the following paragraphs. Generative migration without an AF may be possible, yet, is not the focus of this thesis.

2.3.1 Principles of Generative Migration

The process of generative migration is illustrated in Figure 2.1. It involves using a configuration of building blocks to describe the compositional structure of an agent. Such a description is called a *blueprint*. 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. according to the XML syntax. These descriptions are sent to the destination where an Agent Factory (AF) processes them. The AF re-builds the agent given the blueprint. 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.

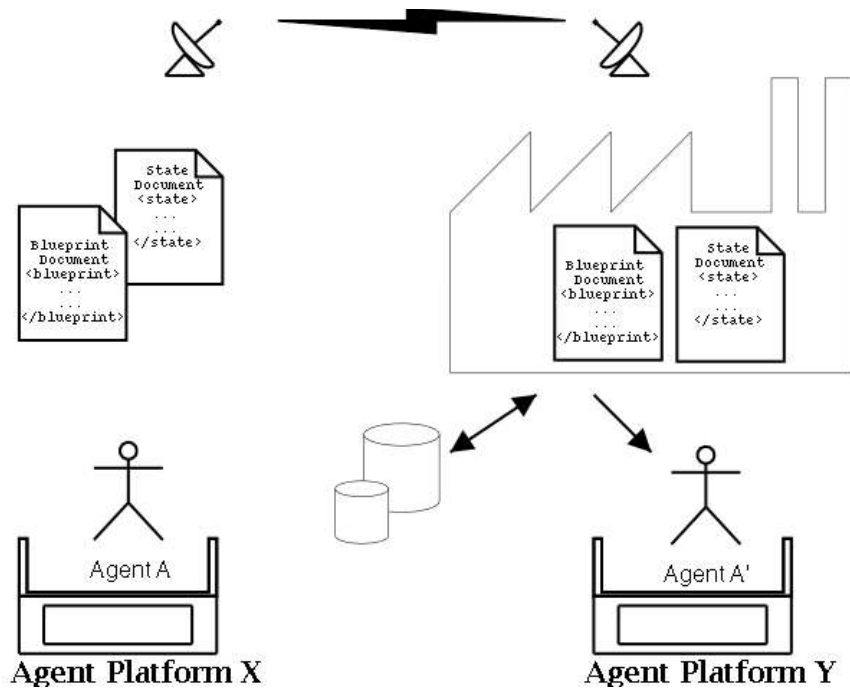


Figure 2.1 Principles of Generative Migration

As can be found in earlier work, (Brazier and Wijngaards, 2002; Brazier, Overeinder, Van Steen and Wijngaards, 2002a; Assem, 2003), using the Agent Factory for generative migration is based on a number of assumptions.

Assumption 1:

Agents have a compositional structure

A compositional structure facilitates the possibilities of adding, removing and changing building blocks of an agent.

Assumption 2:

Components and the compositional structure should be described in a (semi-) formal format, independent of an agent's implementation

The compositional structure should be described in a (semi-) formal format, independent of an agent's implementation.

Assumption 3:

One or more libraries of re-usable components need to be available

The components of an agent are called *building blocks* (abbreviated to BBs). BBs should be re-usable, if BBs are only suitable for the agent for which they are developed, this approach does not offer an advantage over traditional software engineering. BBs should be simple to instantiate for use in newly (automatically) generated agents. BBs need to be stored and retrieved, in a so-called local repository or a library.

Assumption 4:

Two levels of descriptions are distinguished: a conceptual and an operational level

To model agents (independent of the implementation and in an automated process) and to reason about the models it is useful to make distinctions in the levels of detail in the design process. Blueprints and BBs 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, e.g. in UML (Fowler and Scott, 2000). Such a conceptual description may have one or more operational descriptions. A configuration of operational BBs together defines a realization (implementation) of a conceptual model. Separation of the conceptual and operational levels allows more flexible (re)-design and implementation.

2.3.2 Frameworks

To facilitate modelling and execution of component-based agents according to the above-mentioned principles, different frameworks have been developed. Two frameworks are introduced: one conceptual framework and an operational framework.

ICAMP

Scholten, in cooperation with Van Splunter (Scholten, 2003; Splunter, 2002), introduced the ICAMP, the IIDS Conceptual Agent Modelling Paradigm. It is based on the DESIRE modelling framework ((Brazier, Jonker and Treur, 2000; Brazier, Jonker, Treur and Wijngaards, 2001; Brazier, Jonker and Treur, 2002) 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 ICAMP only component structure elements can contain other structure elements. Although ICAMP is based on DESIRE, it differs on some important points; see van (Assem, 2003) for more information.

IAOF

On the operational level, an implementation-language specific framework is needed. For BBs implemented in the programming language JAVA, Scholten en Van Splunter introduced IAOF: IIDS Agent Operational Framework.

IAOF has a one-to-one mapping with the conceptual framework ICAMP; thus, modelling of an IAOF agent is done according to the corresponding concepts in ICAMP. A compositional agent in IAOF consists of:

- A template for a Generic Agent Model
- Building Blocks
- Information Links
- Information Types

In short, IAOF is an operational framework and includes an advanced class-loader for the execution of configurations of compositional JAVA-based agents. IAOF executes configurations of components; one configuration in IAOF represents one agent.

Agent Smith:
Tell me, Mr. Anderson, what good is a phone call
if you are unable to speak?

[The Matrix]

Chapter 3 Interoperability of Agent Platforms

Before diving into the intriguing subject of interoperability, first, a general introduction to agent platforms is given. Followed by an explanation of the three agent platforms that were used for the research of this thesis. Next, a theoretical part on interoperability of agent platforms follows. The chapter presents and explains the assumptions for cross-platform generative migration.

3.1 Agent Platforms

Agent platforms offer 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, pausing, 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 thesis.

Over the last decade numerous Agent Platforms have been developed, each with their own specialties and peculiarities. To name a few: JADE, FIPA-OS, AOS, ZEUS, KADOMA, NOMADS, ARA, AGLETS, GRASSHOPPER, TRACY, AJANTA, LEAP, JACK, SeMoA. This list is not at all complete and just gives a few important agent platforms that had our attention lately (the AgentLink² website currently lists over a hundred available platforms).

Several attempts to compare the platforms took place, for example in (Collier, O'Hare, Lowen, Rooney, 2003; Broos, Dillenseger, Dini, Hong, Leichsenring, Leith, Malville, Nietfeld, Sadi and Zell, 1998; Giang and Tung, 2002).

Most of the articles evaluate a subset of the available features and do their comparison from a different perspective, thus making a general conclusion practically impossible. No absolute prizewinner can be announced.

For this MSc Thesis three agent platforms are studied: AOS, JADE and FIPA-OS. The used versions are: AOS version 0.43, JADE version 3.01b and FIPA-OS version 2.10. A summary of these three agent platforms follows here.

3.1.1 AOS

The AGENTSCAPE project is the current focus of research within the IIDS group. The project includes an agent platform, the AgentScape Operation System (AOS), a number of services including a Generative Migration Facility, and support for application developers. The main goals of the AOS with respect to interoperability are:

² <http://www.agentlink.org>

1. To provide a platform for large-scale agent systems;
2. To support multiple code bases and operating systems;
3. To support interoperability with other agent platforms.

The major challenge in the AGENTSCAPE project is to realize a scalable, secure, and fault tolerant system, that supports multiple distributed applications, heterogeneity, and multiple qualities of service. The AOS provides a platform in which mobile autonomous processes, the agents, can be managed.

The nomenclature of the AOS is as follows. A *location* is a place in which agents and objects can reside. A location in the distributed system consists of one or more hosts. Each host runs a minimal AOS kernel, zero or more agent servers, object servers and service access providers. See also Figure 3.1.

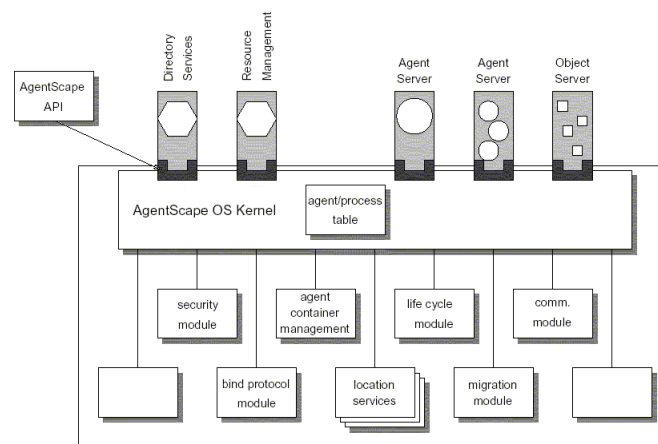


Figure 3.1: AgentScape OS Architecture

Agents are active entities that can interact with each other by means of message-passing communication. *Objects* are passive entities that are only engaged into computations reactively on an agent's initiative.

The current architecture for this system is a middleware layer over sets of network-connected hosts. It is, in fact, a virtual machine distributed over a wide-area network consisting of heterogeneous hosts. On top of that the agent platforms and agent applications can be run.

The current prototype (version 0.43) of the AOS implements the basic functionality required. It is implemented in both JAVA and Python, using XML-RPC for inter process communication. The prototype is available for research purposes, yet, is intended to become an open source project.

This summary of the AGENTSCAPE OS project is based on the articles (Brazier, Mobach, Overeinder, Splunter, Van Steen and Wijngaards, 2002; Brazier, Mobach, Overeinder, Posthumus, Splunter, Van Steen and Wijngaards, 2002; Wijngaards, Overeinder, Van Steen and Brazier, 2002).

3.1.2 JADE

JADE (Bellifemine, Poggi, and Rimassa, 2000; Bellifemine, Poggi, and Rimassa, 2001; Bellifemine, Caire, Poggi, and Rimassa, 2003) stands for Java Agent Development Environment and is free software distributed by TILAB, the copyright holder. Since

May 2003, a JADE Board has been created that supervises the management of the JADE Project. Currently, the JADE Board lists 3 members: TILAB, Motorola, and Whitestein Technologies AG.

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 is available as open source software under the terms of the LGPL (Lesser General Public License Version 2) and can be downloaded from the JADE homepage³. The actual system consists of two parts that are closely related. The first part is the runtime environment for FIPA-compliant multi-agent systems. Secondly, JADE offers a JAVA framework for developing FIPA-compliant agent-systems. The version used for this research is JADE version 3.01b.

JADE has successfully participated both at the first FIPA interoperability test in Seoul on January 1999 and at the second FIPA interoperability test in London on April 2001. The JADE system is based on the FIPA specification and FIPA-compliant (Fipa Inform!, 2001). Furthermore, the JADE team actively participates in the FIPA organization, being a member of the FIPA Architecture Board.

The JADE Agent Model

The motivation for the software architecture of an agent platform can lie in some underlying agent theory. For example, if *beliefs*, *desires*, and *intentions* are built-in expressions, the underlying model is probably the BDI model. The JADE developers made a model of their own based on specific agent properties. The following properties are taken into account:

- Agents are autonomous: this requires each agent to be an *active object* with at least one thread of its own. Having a thread gives the agents possibilities for pro-active behavior (starting a conversation), to make plans and pursue goals.
- Agents are social: communication functionality has to be offered and agents should be allowed to engage in many conversations simultaneously.
- Messages are based on speech-acts: the FIPA standard for Agent Communication Language (FIPA ACL) is based on the so-called speech-act theory and ‘performatives’. See also (Dignum and Greaves, 2000), they explain that performatives are actions that affect the state of the world in the same way physical acts affect the world. In the case of BDI agents (agents based on the notions of *beliefs*, *desires*, and *intentions*) communicative acts are performed in service of intentions, just like any other act. Unlike physical acts, however, the primary effect of a communicative act is to change the beliefs of the parties involved in the communication. According to JADE developers this suggests asynchronous message passing (e.g. using mailboxes for each agent) as a way to exchange information.
- Agents can say “no”: since in a multi-agent system the sender and receiver are equal (as opposed to client / server systems where the receiver is supposed to obey the sender). An agent should be allowed to ignore a received message and even to deny a request. This is closely related to the autonomous property of agents.

³ <http://sharon.cselt.it/projects/jade/>

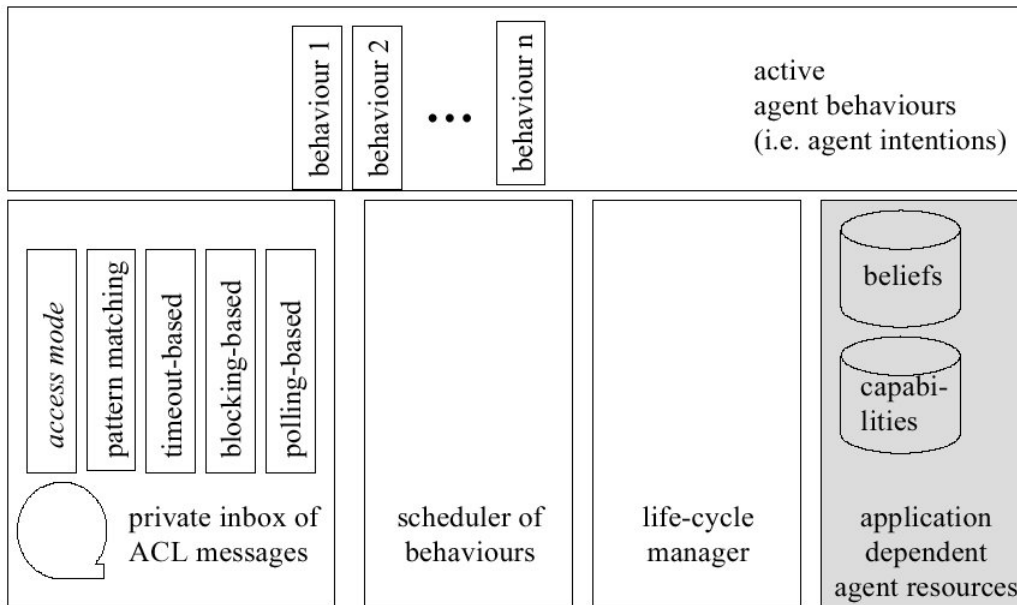


Figure 3.2 The internal JADE agent

Apart from these properties a JADE agent acts according to a pre-specified set of so called Behaviours. See also Figure 3.2. The figure should be read as follows. The Behaviour abstraction models agent *tasks*. A collection of Behaviours are scheduled and executed to carry out the agent duties. JADE provides ready made Behaviours for the most common tasks, such as sending and receiving messages. It is possible to structure complex tasks as aggregations of simple Behaviour patterns.

A distinctive feature for developing reasoning agents is the JessBehaviour. It allows integration with the JESS (JAVA Expert System Shell)⁴, a package for rule programming. JESS uses the RETE Algorithm for processing facts and rules. Literature on JADE does not explain the exact role of the ‘life cycle manager’ and ‘application dependent agent resources’.

JADE Platform organisation

The organizational structure of JADE is illustrated in Figure 3.3. Basically, JADE can be run on a single machine or split over several hosts (physical machines). However, acting as a truly distributed system, JADE appears to the outside world as a single entity (communication to the outside world goes via the *Main Container*). A JADE system consists of one or more *Agent Container* instances, each one running in a separate Java Virtual Machine. The *Agent Container* is comparable with the notion of *location* in other agent platforms. In JADE, agents can migrate between connected containers.

⁴ <http://herzberg.ca.sandia.gov/jess/>

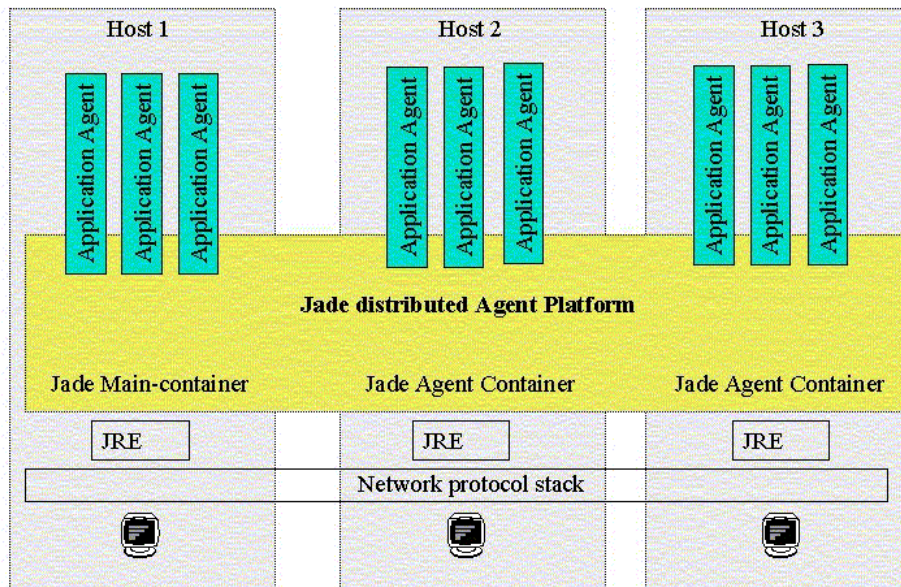


Figure 3.3 The software architecture of the JADE Agent Platform

The communication architecture of JADE offers flexible and efficient messaging by means of a number of protocols (Bellifemine, Poggi, and Rimassa, 2000). Communication is possible via JAVA RMI, IIOP and HTTP. JAVA RMI is used to communicate among the Agent Containers (for example for passing ACL-messages of agent-to-agent communication). IIOP and HTTP are used for communication with other platforms. Each agent container can act as an IIOP client to forward outgoing messages. A special Main Container has the role of IIOP and HTTP server, listening at the agent platform ACC-address for incoming messages from foreign platforms. Despite the possibilities for cross-platform message exchange, in the default releases of JADE agents cannot migrate between platforms.

3.1.3 Fipa-OS

FIPA-OS⁵ is an open source agent platform implemented in the JAVA programming language (Poslad, Buckle, and Hadingham, 2000; Poslad, Buckle, and Hadingham, 2001). FIPA-OS version 2.10 is used in this project.

Originating from the Nortel Networks Company, Nortel Networks' Agent Technology Group formed an independent start-up company, called emorpha. Their goal is to apply leading edge software and agent technology to commercial development of smart wired and wireless Internet services.

The main purpose of FIPA-OS is to provide a reference implementation for the standard specifications of the FIPA organization. The platform supports communication between agents using the FIPA Agent Communication Language (FIPA ACL) specification. 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. Figure 3.4 illustrates the core components of the FIPA-OS agent platform.

⁵ <http://www.emorpha.com/research/overview.htm>

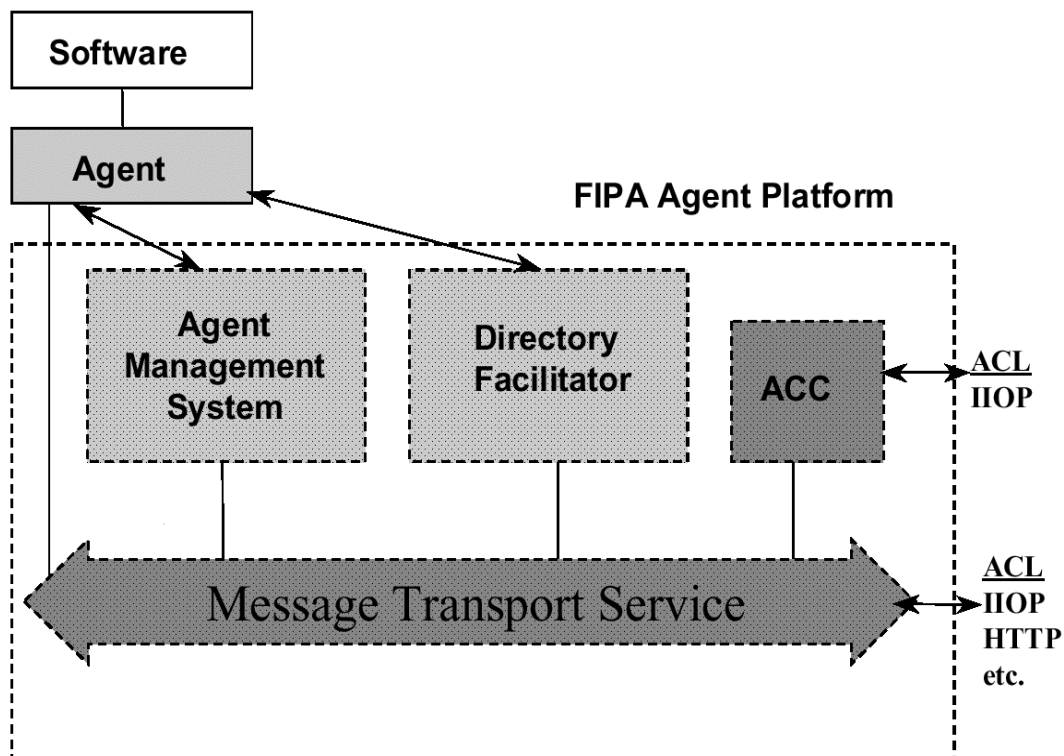


Figure 3.4 FIPA 97 Reference Model

The lowest level is the Message Transport Service, which connects the components of the platform and agents on top of it. Messaging with other FIPA-OS platforms is supported over the protocols IOP and HTTP. HTTP can also be used for connecting to other types of agent platforms or for interaction with regular programs. Special services in the platform are the Agent Management System (AMS), the Directory Facilitator (DF) and the Agent Communication Channel (ACC). By default, agents are assumed to communicate via the ACC, but FIPA does not prohibit agents from communicating directly with each other.

Fipa-OS is being deployed in several application domains including virtual private network provisioning, distributed meeting scheduling and a virtual home environment. It has been demonstrated to interoperate with other FIPA compliant platforms, that is, it can exchange messages with JADE and ZEUS. Note that the default releases of FIPA-OS do not support agent migration.

3.1.4 Agent Platform Topologies

In general, a distinction can be made between distributed agent platforms and single node (or non-distributed) agent platforms. See also Figure 3.5.

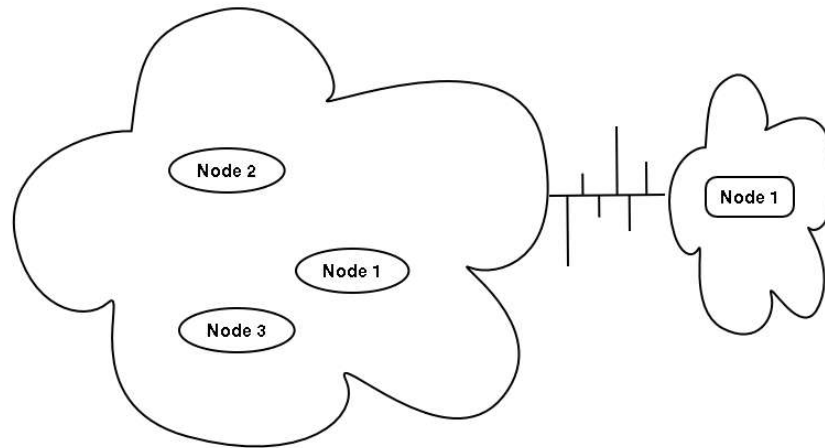


Figure 3.5 Agent Platform Topologies

As depicted in the left cloud in the figure, a distributed agent platform couples a number of nodes that are network connected. Each of such nodes can hide a number of computers (also called hosts), on each of the computers the agent platform software has to be installed and running. In such an environment it is possible to migrate agents between the various nodes of the agent platform. Examples of distributed agent platforms are the agent platforms JADE and AGENTSCAPE OS.

Some agent platforms do not offer the possibility of coupling nodes. They form the category of 'Single Node' agent platforms do mostly not support mobility. An example is the FIPA-OS agent platform.

3.2 Interoperability

Several definitions of interoperability found in the literature are presented and evaluated here. In the end of this section, a new definition is presented.

A definition of interoperability with respect to Distributed Systems in general, by (Tanenbaum and Van Steen, 2001) is as follows:

Interoperability: "Interoperability characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified in a common standard."

Related to that is the definition of portability, also from (Tanenbaum and Van Steen, 2001):

Portability: "Portability characterizes to what extent an application developed for a distributed system A can be executed without modification on a different distributed system B that implements the same interface as A."

These definitions are quite general, however they do point in the right direction. The interoperability definition clarifies that it is important to describe the services and the way they interact in a common standard (a specification document). When considering

agents as application-special parts of an agent platform, the portability definition points at the fact that interfaces should be the same (possibly pre-defined as well).

A quite general definition of interoperability with respect to agent systems comes from (Magnin et al, 2002a): "allowing agents to run, communicate and move between agent platforms". And in (Grimstrup et al, 2002) the following was found: "allow foreign agents to execute in a non-native mobile-agent system".

Finally, a more specific definition on interoperability of mobile agent systems is given in (Roth, Pinsdorf and Binder, 2001):

Interoperability: two mobile agent systems are *interoperable* if a mobile agent of one system can migrate to the second system, the agent can interact and communicate with other agents on this system (or even remote agents), the agent can leave this system, and it can resume its execution on the next interoperable system.

Other definitions of interoperability take the form of standard specifications, most notably the FIPA and MASIF specification, a summary of both follows. These specifications form the 'common standard' as mentioned in the interoperability definition by Tanenbaum and Van Steen.

FIPA

In 1997 the first standard specification for Multi-agent systems was proposed by the Foundation for Intelligent Physical Agents (FIPA). FIPA is an international non-profit organization and its goal is to produce specifications for generic agent technologies. They publish updates and new specifications ever since. These specification documents specify the normative rules that allow a society of agents to inter-operate. Effectively it means that they specify a model for the agent environment, a life cycle model and support for communication and agent management.

The FIPA specification describes a reference model for agent platforms and identifies the roles of some key service agents:

- AMS: Agent Management System is the agent that exerts supervisory control over access to and use of the platform; it is responsible for maintaining a directory of resident agents and for handling their life cycle;
- ACC: Agent Communication Channel provides basic contact between agent inside and outside the platform.
- DF: Directory Facilitator is the agent that provides a kind of 'yellow page' service to the agent platform.

Although the JADE literature (Bellifemine, Poggi, and Rimassa, 2000; Bellifemine, Poggi, and Rimassa, 2001) refers to these entities as agents, we would rather call them services. Other agents use them, they provide means for communicate and perform tasks. In other words, they provide services. Secondly, they do not fit into the agent model because a true agent would adhere to the autonomy principle, thus being able to deny a request. It is not the purpose of the specification that any service would be denied at the liking or disliking of a stubborn agent.

The FIPA specification also defines the Agent Communication Language (FIPA ACL), which is used to exchange messages.

Another issue with respect to FIPA is compliance; the FIPA organization has not yet specified an official procedure to test compliance of implementations. Anyone can claim that his or her implementation complies to the standard. More information on the FIPA specifications can be found at the website of the FIPA organization⁶.

OMG MASIF

The Object Management Group (OMG) proposed their Specification for Mobile Agent System Interoperability Facilities (MASIF, sometimes also abbreviated to MAF, Mobile Agent Facility) a little later the first FIPA-specifications (Milojicic, et al., 1998). Its purpose is to provide a specification for interoperability between mobile agent platforms of different vendors. The MASIF specification document covers terms and conventions, such as mobile agent, agent names, place, region, etc. Furthermore it defines a set of standard interfaces and data types, the basic organizational structure of a mobile agent platform and a minimal communicational architecture. According to the Mobile Agent Facility Specification the mobile agent community should standardize the following to promote interoperability: Agent Management, Agent Transfer (migration), Agent Naming and Agent System Naming, Agent System Types, Location Syntax. In (Milojicic, et al., 1998) states that “*Agent Communication* is outside the scope of MASIF, and it is extensively addressed by CORBA”. However, to our knowledge CORBA covers how objects are shared and communicated and not the format of the message objects (e.g. ACLMessages according to the FIPA ACL specification).

The Grasshopper Agent Platform⁷ (Bäumer, Breugst and Choy, 1999) attempts to comply with the MAF Specification, though its goal is not a reference implementation for the specification, it was the first to recognize the standard.

Unfortunately the MAF Specification is not precise, incomplete and interpretable in different ways, which could lead to different non-interoperable implementations. Besides that, the MAF Specification did not get enough recognition from the research communities to be an internationally recognized standard.

Definitions and standard specifications, combined with practical experiences with agent platforms lead to a distinction of two different types of interoperability:

1. Message Interoperability: if two agent platform systems can exchange messages, implying that the agents residing on it can also exchange messages.
2. Migration Interoperability: if agents can move between the two systems and successfully execute on both sides.

To cover details of both types of interoperability standard specifications are needed, defining a common set of terms and details on the ‘what and how’ for implementations. For example, definitions are needed for the services, standardized interfaces and the formats of the data-exchange. An example implementation and a set of standard tests should validate interoperability and compliance to the specification.

Currently, message interoperability is covered in the FIPA Specifications: agents talk a standardized ACL and communicate to the platform that uses HTTP and IIOP (via the CORBA ORB) protocols to handle the message transport. Also, the content languages and

⁶ <http://www.fipa.org>

⁷ <http://www.grasshopper.de>

message sequences of messages are being addressed in respectively Ontology and Conversation specifications. FIPA attempted to define migration interoperability in their “FIPA Agent Management Support for Mobility”-specification, however, immediately gave it a deprecated status. To our knowledge, the exact reasons for the deprecation of the specifications are not known publicly. As is explained in Chapter 2: Agent Migration, FIPA does not support migration interoperability (because they do not define the APIs or standard interfaces).

The MASIF specification addresses migration interoperability, however, does not treat message interoperability. Neither FIPA nor MASIF standardizes security, though both claim it is important and the issue needs attention in the near future. The results of this comparison are summed up in Table 1.

Subject	FIPA	OMG MASIF
Agent Management	Yes	Yes
Agent Communication	Yes	No
Agent Mobility	No	Yes
Security	No	No

Table 1 FIPA versus OMG MASIF

3.3 Cross-Platform Generative Migration

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

- 1) The appropriate communication facilities need to be present in the agent platforms: e.g. 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 the migration protocol and sending/receiving information about the agents (e.g. blueprint and state documents)

- 2) Presence of 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 would start and initialize a new agent.

- 3) 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. The Java Virtual Machine offers a level of abstraction over the operating system, thereby creating an OS-independent programming language: JAVA.

Without the restriction of the programming language additional difficulties are

introduced. For example, new building blocks in the target language are necessary for the target platform (possibly with help of automated code translation), or the target has to be adapted in such a way that it can handle JAVA building blocks.

The agent platforms AOS, JADE and FIPA-OS satisfy the third condition. The test-results of the first two assumptions are presented in Table 1.

		Communication Protocols	Creation Process
JADE		HTTP, CORBA ORB (IIOP)	simple, can be isolated
Fipa-OS		HTTP	currently part of the GUI, difficult to isolate
AOS	0.43	(internal only)	Service in the platform
	1.0	HTTP	Service in the platform

Table 2 Test results

The test results show that communication in JADE and FIPA-OS is currently possible over the HTTP protocol. Version 1.0 of the AOS supports the HTTP communication protocol, yet, is not available at the moment.

Note that coupling JADE and FIPA-OS to exchange messages is not a trivial task. 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.

As can be seen in the table, 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. However, the creation process is available in JADE and AOS.

Combining these results, a practical approach is developed in this thesis for Generative Cross-platform Migration from FIPA-OS to the JADE agent platform.

Trinity: Neo... nobody has ever done this before.
Neo: I know. That's why it's going to work.

[The Matrix]

Chapter 4 Practical Work

This section on the practical work is divided in two subsections. The first covers the design of the demonstration and the second the implementation to convincingly demonstrate cross-platform generative migration.

4.1 Design

The design of the demonstration combines traditional Software Engineering methods and agent design. Scenario writing, UML Class and Interaction Diagrams (Fowler and Scott, 2000) stem from Software Engineering and modelling of the internals of the agents is done according to concepts of ICAMP (IIDS Conceptual Agent Modelling Paradigm), based on the DESIRE design methodology (Brazier, Jonker and Treur, 2000; Brazier, Jonker, Treur and Wijngaards, 2001; Brazier, Jonker and Treur, 2002).

4.1.1 Scenario's

One or more of the example scenarios for heterogeneous inter-platform migration (cross-platform migration) is to be implemented. For example, a demonstration that shows migration of an agent from one platform (e.g. AGENTSCAPE OS OR FIPA-OS) to another (JADE).

Scenario 1: Insurance Company Application

The setting for this scenario is an application that handles interaction between a health insurance company 'AllSureThings' and a hospital 'HaveYouBeenComfortable'. In the application a procedure takes place that is normally handled by two human operators: a claim-handler of the insurance company and a doctor (or the assistant /secretary representative). The scenario is based on the typical situation that a client of the insurance company, Mr. B, has been treated by a physical therapist at the hospital and requests that the insurance company pays the bills. Before payment can take place, the insurance company needs to check the validity of claim. A similar case, in the domain of International Traffic Insurance, can be found in (Aart, Marcke, Pels and Smulders, 2002).

The insurance company runs an application built on the agent platform of type A (e.g. AgentScape) on which an agent runs. This could be an agent like the information retrieval agent described in (Brazier, Splunter and Wijngaards, 2001). In the claim-check application the data about Mr. B's claim is handed over to the agent, together with an assignment to match the given data to the available data at the hospital.

This agent then requests to be moved to the agent platform of the hospital. From the hospital platform access to specific local resources is regulated, such as database access.

As often happens in the real world, both organizations run different software. In this case the hospital only receives agents at a JADE agent platform.

Currently, the agent is an AgentScape agent and the internal model of the agent is designed according to the Desire Generic Agent Model (Brazier, Jonker and Treur, 2000). The JADE agent platform requires a JADE agent. Thus, a translation process has to take place to generate a new agent that can be run on the remote JADE agent platform. This is where the Agent Factory (AF) comes into play. The AF is requested to perform the task of re-design and produce an agent according to the specific (new) requirements (JADE agent, JADE internal model).

In this case a blueprint for the agent needs to be available as well as building blocks for AgentScape- and JADE-agents. Additionally, solutions for state saving and transfer of state need to be available as well. Given these preconditions the Agent Factory on the receiving side regenerates the agent where it continues execution.

After an agent's task is completed at the hospital, a next step would be to either communicate the results back to the application or to migrate back home in a similar fashion.

Note that normally an agent or any outside application would NOT be allowed to have access to the database of any hospital. As in this situation the agent is regenerated locally and consists of trusted components, this security issue is circumvented.

Scenario 2: Remote Computation Agents

The setting for this scenario is an application for remote computations, given the ideas of remote compute servers performing calculation tasks on request of a user. The Java Tutorial on RMI was the source of inspiration for this scenario; see the Client/Server Trail on RMI (Wollrath and Waldo, 1998). The server in the example implements a generic compute engine, which the client uses to compute the value of π (pi)

A user may have an application that can perform heavy and complicated calculations, for example, a spreadsheet or scientific math program. However, the user's local workstation or PDA may not be up to the task at hand, or the user may also want to continue his/her tasks. Therefore the application is embedded in an agent platform environment such as AgentScape. The task of the computation is handed over to an agent and the agent is instructed to leave the local machine before starting the computation task.

This agent looks up a possible location to which to migrate, for example a remote server that supports heavy computations. The agent issues a migration request to the local migration service in the AgentScape platform. The migration service contacts the location and discovers that the target server only runs JADE-agents. Since this agent is an AgentScape-agent, the agent will have to be 'translated'. Just like in the first scenario, this is where the Agent Factory (AF) comes into play. The AF is requested to perform the task of re-design and produce an agent according to the specific (new) requirements (JADE agent, JADE internal model).

After the Agent Factory has assembled an agent, an agent is initiated with its state. Then, the agent is launched into the platform where it continues execution.

Scenario 3: Technical Scenario

This scenario focuses on the technical aspects of generative agent migration. It describes the parties involved and the interactions between the parties.

The parties involved are: two Agent Platforms (AP_Q of type X and AP_P of type Y), agent Z (that requests the migration), the Directory Facilitators of the both platforms and the Generative Migration Services on both sides. The X and Y variables for agent platform types can be instances of agent platforms e.g. AgentScape OS, Fipa-OS or JADE.

The Generative Migration Services take care of migration, they are registered at the local DF and can be found by agents and other services because the DF's are cross-registered.

Thus, an agent can send a migration request to the local GMS for migration to a remote platform. The local GMS contacts the remote GMS and forwards the request, if the remote platform has a GMS migration can take place. An agent should receive an acknowledgement about this, or denial with an indication of the reason for denial (e.g. no GMS on target location, building-blocks not present on target or agent with identical name already 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 old agent in the source platform can be stopped and removed.

4.1.2 Interaction Diagram

Interaction Diagram 1 illustrates the Technical Scenario as described above and is found in Appendix I. In the interaction diagram a number of actors are distinguished:

- Human User: The human user initiated the scenario by starting the agent Agent Z on Agent Platform P
- Agent Z: the agent that looks-up the GMS in the Directory Service and requests migration.
- Directory Service (DS): this service provides information on other agents and services. At least one DS is needed, running on Agent Platform P so the Agent Z can find the Generative Migration Service.

The service for migration, the Generative Migration Service (GMS) is split-up in two functional parts, each performs a sub-set of the total of tasks that have to be handled for the migration process:

- GMS@AP_P: the Generative Migration Service on Agent Platform P. This GMS receives a request from Agent_Z to move to a specific target platform: Agent Platform Q. The GMS is assumed to have the contact address of the GMS that resides at Agent Platform Q.
- GMS@AP_Q: the Generative Migration Service on Agent Platform Q. This GMS receives a move request from the GMS on Agent Platform P to move Agent Z to this agent platform. It acknowledges the request and receives the blueprint and state-data of the agent. Based on that information a new Agent Z is

created on Agent Platform Q: Agent Z'.

The Generative Migration Service (GMS) is a central service in this scenario. The GMS is a platform specific service that performs a number of tasks:

- Handling the agent migration protocol;
- Receiving blueprint and state data from an agent;
- Forwarding the blueprint and state data to the GMS at the target agent platform;
- Building a new agent given the request, blueprint and state data;
- Launching the agent into the platform such that the agent's reasoning process in started.

Several alternative approaches are possible with respect to the scenarios:

- 1) Agents could take care of the state transfer themselves if the original agent is reactivated after its sleep with the special purpose of communication. It is also possible that the agent does not at all go to sleep and e.g. takes also care of buffering messages until the new agent is running. This alternative scenario is depicted in Interaction Diagram 2 in Appendix I.
- 2) The GMS can be a centralized service that is registered at both platforms, or be present in both platforms. However, it needs the same capabilities. In the technical scenario one GMS would be sufficient if only migration to the target platform is required. A delete-request for the migrating agent should then be issued to the AMS of the source platform, or the agent has to request the delete.
- 3) Agents need to find the appropriate services to be able to request a migration; introducing a Directory Service where the agents can look up the services solves this issue. Cross-registration of Directory Facilities (the FIPA Directory Service) is possible when the involved locations are based on the same standard. However, connecting and exchanging information may be complicated if this is not the case e.g. coupling JADE's DF and the Location Service of OAS.

4.1.3 Class diagram for IAOF integration

In this project the IAOF is integrated in the JADE agent platform. The IAOF (IIDS Agent Operational Framework) makes it possible to use agents consisting of configurations of building blocks in JAVA-based agent platforms. See Figure 4.1 UML Class Diagram for integration of IAOF.

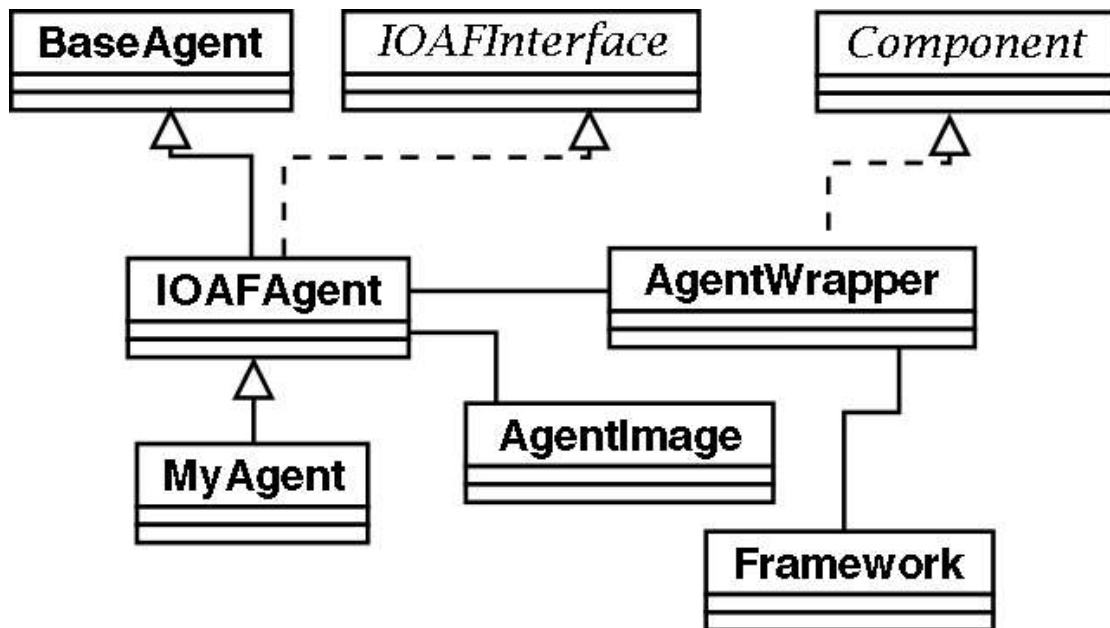


Figure 4.1 Class Diagram for IOAF integration

The first class to be considered is in the upper left corner of the diagram: the `BaseAgent`. In most JAVA-based agent platforms the developers of the platform deliver a `BaseAgent` class. The `BaseAgent` has to be extended in order to make one's own agent. For example, in FIPA-OS the `BaseAgent` is called `FIPAOSAgent` and every other agent in the platform is an extension of it.

The goal of the design is standardization of the IOAF in all agent platforms that will support IOAF. Therefore an interface is defined, the `IOAFAgentInterface`, and the `BaseAgent` is extended so the interface is implemented as well:

```

public class IOFAgent extends BaseAgent implements
IOFAgentInterface{
    //code for IOFAgent
}
  
```

Then, if one wants to make his own agents, and use the facilities that are provided by a `IOFAgent`, the only thing needed is to extend the `IOFAgent`:

```

public class MyAgent extends IOFAgent{
    //code for MyAgent
}
  
```

So, why are the `IOFAgent`, the `AgentWrapper` and the interface necessary? Well, the interface is not strictly necessary. However, it is better and tidier object-oriented programming. The `IOFAgent` and the `AgentWrapper` are really necessary since we want to use the `IOAF Framework` class for execution of an agent configuration in such a way that it is easy to make other IOAF agents. A `BaseAgent` extension could directly use the `Framework` class, however, that does not allow for easy re-use since in every new agent the same code has to be copy-pasted. Also, our current design is according to the same design for IOAF integration in AOS.

To use the `Framework` class we need to instantiate it with an `AgentImage`, which is a storage unit to hold all configuration files, the complete building block configuration for an agent, information types and a state representation.

So, the `AgentWrapper` wraps the `Framework`, instantiates it with an `AgentImage`, and starts the reasoning process. Calls from the reasoning process to the `AgentWrapper` are forwarded to the `IAOFAgent`, where they are translated in to calls to the methods of the `BaseAgent`. This design resembles the “Mediator Pattern” from *Design Patterns* (see Gamma, Helm, Johnson and Vlissides, 1995), where method calls from a class A are translated by a class B to calls to a class C.

The design as presented here should be generic in such a way that it can be re-used in other JAVA-based agent platforms.

4.2 Implementation Problems

During implementation a number of problems were encountered as described in this subsection.

4.2.1 Translating information

While programming the `IAOFAgent` and `AgentWrapper` for JADE it became clear that translation of platform specific information to the IAOF Information Types (ITs) would not be an easy job. However the solution was to do the translation in the sub-components of the `AgentWrapper`. There, platform specific information is translated to ITs (or vice-versa) during their activation (reasoning rounds):

- 1) `ManageSentMessageQueue`: For every `ToBeSentIT` message, it calls to the `AgentWrapper` method to send the message;
- 2) `ManageReceivedMessageQueue`: Converts ACL messages to `MessageIT` messages and puts them on the output of the `ManageReceivedMessageQueue`-component;
- 3) `DirectoryServices`: takes care of constructing an `OwnAgentIdentifierIT` to allow an agent to know who and where it is. Also information on all agents currently residing on the same platform is provided in the form of a `PresentAgentIT`.

4.2.2 The `AgentImage`

The `AgentImage` class is a storage unit to hold all information related to an agent formatted as a ZIP-file in memory. It can contain configuration files, the building blocks for an agent, information types and state representation objects.

In AOS the `AgentImage` of an agent used to be created via a Python start-up script. In the demonstration implementation of this thesis, creation of the `AgentImage` had to be done in JAVA. Writing a JAVA method that added files to the image wasn't trivial since the `AgentImage` API did not directly support that function. The solution to the problem was to read a file from a `FileInputStream` into a `ByteArrayOutputStream` and add the `ByteArray` to the `AgentImage`.

Another problem was the setting of the name of the entries in the `AgentImage`. Apparently the relative file path was discarded and files (or other stored objects) were identified only with a name (filename or key).

These details are covered up with a method that can add a file to the `AgentImage` given a filename, a relative file path (directory specification) and a storage key.

4.2.3 Package Integration

Originally, the implementation was intended as an extension to the JADE agent platform using the libraries (JAR-files) of IAOF. In a separate subdirectory classes were coded that imported the IAOF classes. Each new IAOF-JADE agent would be placed in a new subdirectory and actually the whole extension became rather messy and awkward to work with.

As an alternative, the produced material (`IOAFAgent`, `AgentWrapper` and the subcomponents of the `AgentWrapper`) could also be regarded as a JADE-special extension of the IAOF package. If integrated in the IAOF package it has advantages for a user who wants to use it. For example only one library (the IAOF JAR-file) is needed for development of agents that can run on all supported platforms (currently AOS and JADE). Additionally, the logical structure of the package is preserved.

Neo: Who are you?
Bane/Agent Smith: Still don't recognize me? I admit it is
difficult to even think encased in this rotting piece
of meat. ... How pathetically fragile it is ...

[The Matrix Revolutions]

Chapter 5 Related Research

The previous chapters explained how generative migration could be used for cross-platform agent migration. This chapter describes research that is related to the research in this thesis. First, other agent factories are presented and compared with the IIDS Agent Factory. Related to the research in this thesis and to the Agent Factory research is the subject of agent modelling. The last part describes research projects on interoperability and cross-platform migration of agents. A comparison between our approach and theirs is presented as well.

5.1 Agent Factories

In this thesis, an Agent Factory is a service to design, adapt and (re)assemble agents from building blocks (Brazier and Wijngaards, 2002; Brazier, Overeinder, Steen and Wijngaards, 2002b; Splunter, Wijngaards, Brazier, 2003). It is being developed in the IIDS research group, therefore it is called the IIDS Agent Factory. Besides the IIDS Agent Factory some other Agent Factories have been developed, these are presented in the following short summaries and compared at the end of the subsection.

5.1.1 Agentfactory.com

The Agent Factory⁸ has been developed as part of ongoing research at University College Dublin, which is concerned with the creation of *"a cohesive framework that supports a structured approach to the development and deployment of agent oriented-applications."* It delivers extensive support for the creation of Belief-Desire-Intention (BDI) agents. The key features of Agent Factory are:

- 1) An interpreted agent programming language that is derived from a formal logical model.
- 2) An Agent Platform that supports the deployment of agents written in this language.
- 3) Lightweight agent implementation that can be deployed on PDAs.
- 4) Prefabricated system agents including FIPA Agent Management System and Directory Facilitator agents.
- 5) Well-defined methodological support for the fabrication of agents.
- 6) An integrated toolset that aids the fabrication process.

The articles (Collier and O'Hare, 1999; Collier, O'Hare, Lowen and Rooney, 2003) explain their project as "The Factory of the Agents".

⁸ <http://www.agentfactory.com>

5.1.2 The Italian Agent Factory

The “Italian” Agent Factory of Cossentino and his group (Cossentino, Burrafato, Lombardo and Sabatucci, 2002a; Cossentino, 2002b) is supported by the AgentCities⁹ organization. They intend to deploy 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 to their existing ones. The patterns are meant to be agents' patterns described as pieces of models and code.

Agents in this project are developed according to the PASSI design methodology, with which models of multi-agent systems and agent interactions are specified and expressed in UML. The pieces of models referred to are structure diagrams expressed in UML class diagrams and the behavioural diagrams in UML activity diagrams.

The service they want to provide to the Agentcities Network's designers is an application that allows them to either create new agents or upgrade their existing ones with services and/or tasks contained in the repository.

As for the creation of a new agent, the designers can choose the functionalities they want to introduce from the repository and the deployment platform among the supported ones (FIPA-OS, JADE) for the code production. As for the upgrading-operation, the input of the application will be code and/or UML models of an agent. The models will be represented in XMI. In both cases, the application will provide some validation mechanisms for the designers' inputs. And finally, the application will return the created or upgraded agent.

5.1.3 The German Agent Factory

The newest technology of the VIRAGE COMPANY¹⁰ enables a webpage to talk to visitors. A virtual employee represents you or your company on the webpage. It is a speech enabled “Chatterbot” that handles product sale and requests for information.

The VIRAGE COMPANY configures these soft-bots from off-the-shelf components. It delivers a commercial product aimed at e-commerce and business applications.

5.1.4 Comparison

The focus of the IIDS Agent Factory project is on automated design and redesign of agents, whereas in the projects Agentfactory.com and the Italian Agent Factory the focus is on assisting users. Also, their focus is on multi-agent systems, where the Agent Factory of IIDS focuses on single agents.

The German Agent Factory is clearly not an Agent Factory in our sense of the word, although it is superficially related since they configure agents from pre-manufactured components. When comparing the above with the IIDS Agent Factory, some similarities can be found in the approaches to agent design and the assembly process.

5.2 Agent Modelling

Various methodologies are available for Multi-Agent System (MAS) development, for example AUML, Gaia, AgentFactory, JATLite, AgentBuilder, Zeus and DESIRE. Some methodologies come with modelling tools and an agent (prototyping)

⁹ <http://www.agentcities.org>

¹⁰ <http://www.agentfactory.de>

environment, while others do not (AUML, Gaia). An interesting article in this context is (Moraitis, Petraki, and Spanoudakis, 2002); it presents an attempt to use the Gaia methodology to engineer a multi-agent system that is to be implemented in the JADE agent platform. Gaia is used for high level design and the paper describes a kind of roadmap for the implementation of a Gaia model in JADE.

In contrast to MAS development methodologies only a few models are available for conceptual modelling of an agent's internals. An example is the DESIRE methodology that includes the Generic Agent Model (Brazier, Jonker and Treur, 2000) used in this thesis. DESIRE consists of conceptual analyses and design methodology and a prototyping environment. Another example of a conceptual internal model is the 5C Model (Five Capabilities Model) as used in an application for international insurance traffic with software agents (Aart, Marcke, Pels and Smulders, 2002). The generic tasks as identified in both models are more or less the same. Hence, a mapping between the two is presented in a brief explanation of the 5C Model.

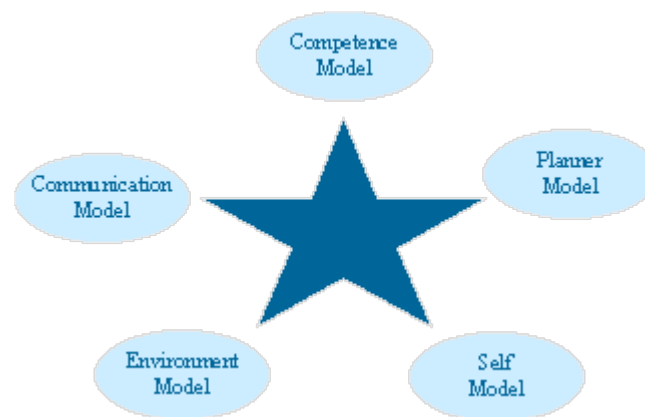


Figure 5.1 Five Capabilities Model

Figure 5.1 depicts the dimensions of the 5C Model, i.e. the competence-model, the communication-model, the environment-model, the self-model, and the planner-model. The *competence-model* contains the methods and knowledge that enables an agent to execute the tasks it is designed for. It is analogous to the task of the Agent Specific Task component in the DESIRE Generic Agent Model (GAM). The *communication-model* is responsible for handling all interactions, with agents and other systems. In GAM this task is split-up over two components: Agent Interaction Management and World Interaction Management. The *self-model* gives an agent an idea of what the agent is doing, e.g. what are its tasks, goals, jobs, states, competences, etc. The *planner-model* enables an agent to autonomously decide how to spend its time. For example, it can contain various planning strategies for meeting the agent's goals. The tasks of the *self-model* and *planner-model* can be placed in the Own Process Control component. The *environment model* finally, gives an agent a view on the world it operates in, e.g. with which other systems and agents can it interact and what information is stored about them. Again, in GAM this task is split-up over two components: Maintenance of World Information and Maintenance of Agent Information. Concluding, it can be noted that for every capability a component of the GAM is available.

Other work related to the above comparison describes how the Generic Agent Model (GAM) can be refined to obtain a formally specified design model (Brazier, Jonker and Treur, 2000). Described are refinements to four other existing agent architectures: TOURING MACHINES, INTERRAP, ZEUS, and ADEPT.

5.3 Alternative Approaches for Interoperability

In the past few years several attempts to enable interoperability and cross-platform migration in agent platforms took place and some of them resulted in published material. A short summary of the research projects follows, after which a comparison with our approach is presented.

5.3.1 Guest Agents

The developers of the *GUEST AGENTS* (Magnin et al, 2002a; Magnin et al, 2002b) propose a middleware-based model that introduces an intermediate layer (the *GUEST LAYER*) for the support of their agents on top of an existing agent platform. Regardless of the platform, if it can support the *GUEST LAYER*, the agents can be executed. The goals of the project are three-fold:

- 1) Allowing interoperability between platforms, i.e. allowing agents to run, communicate and move between them;
- 2) Providing 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) Adding new adaptive features to the platforms, i.e. plug-in mechanisms for dynamic extensions.

Though still an experimental prototype *GUEST* already enables interoperability between *VOYAGER*, *AGLETS*, *GRASSHOPPER* and *JADE*. No source-code or libraries are available for public or research usage.

5.3.2 Monads Project

The Nomadic Computing Group in the Department of Computer Science at the University of Helsinki, Nokia Mobile Phones, Sonera, and Nokia Telecommunications carried out the research project *MONADS*¹¹ (Misikangas and Raatikainen, 2000). The project started in February 1998 and was scheduled to run until December 2000.

The *MONADS* project concentrated on the needs of nomadic users and on 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 software agents and to develop prototypes based on that architecture. The objective of the prototype implementation is to evaluate the functionality of the essential features of the designed architecture.

Dynamic adaptation of a service to the properties of terminal equipment and available communication infrastructure is an attractive feature. Thus agents themselves are not adaptive, only used to steer changes in e.g. quality of service.

¹¹ <http://www.cs.helsinki.fi/research/monads/>

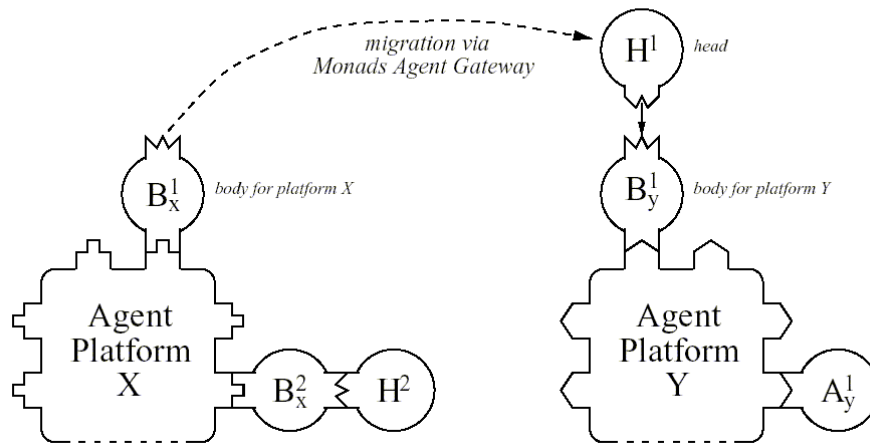


Figure 5.2 Monads Agent Head Migration

The basic approach 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. As depicted in Figure 5.2, the head can migrate via a service called the Monads Agent Gateway (MAG).

Some interesting problems that were met during the project are the following:

A) Platform independent services: solved by using a Java-specific trade using extension of a superclass.

B) Maintaining identities: at migration only the head is migrated, and the body stays alive at the platform. When the head returns to this platform, it is placed on the same body as before, therefore maintaining the same identity of the agent (multiple platforms do not necessarily use the same identifiers).

Other (non-implemented/further researched) options/features given were:

- A body generator: creating a body at arrival on a new platform, given a list of interface classes that should be supported. In our terms a simple agent factory could be used for the generation of the wrapper;
- Template-based service body source code: no need of a generator, but additional data is to be transferred with the head;
- Carrying pre-created and pre-compiled bodies for all agent platforms, all the platforms need to be known beforehand.

The MONADS approach has been implemented on the following agent platforms: VOYAGER, JADE and GRASSHOPPER. Unfortunately, no source-code or libraries are available for public or research usage.

5.3.3 SeMoA

SEMOA¹² stands for Secure Mobile Agents and is concerned with the development of extensible and open servers for mobile agents (Pinsdorf and Roth, 2002; Roth, Pinsdorf and Binder, 2001). The servers and agents are written in JAVA. The focus of the project is on all aspects of mobile agent security, including protection of mobile agents against

¹² <http://www.semoa.org/>

malicious hosts. Another important feature is SeMoA's interoperability with other platforms such as AGLETS, TRACY and JADE, which enables running their agents in a SeMoA server environment. This is done by means of a LifeCycle wrapper, which translates method calls of the foreign platform (JADE, TRACY) to the SeMoA native calls. SeMoA is freely available for download as open source and extensive documentation is available. The license is modelled along the lines of the Lesser GNU Public License (LGPL), modified so that it is compliant with German law.

5.3.4 GMAS

The Grid Mobile Agent System (GMAS) (Grimstrup, et al., 2002), allows mobile agents to migrate to different mobile agent systems and operates on top of the DARPA CoABS Grid. It is a cooperation effort between Dartmouth College, the Institute for Human & Machine Cognition (University of West Florida) and the Lockheed-Martin Advanced Technology Laboratory.

The basic approach is to allow foreign agents to execute in a non-native mobile-agent system by translating the foreign APIs into the local platform APIs. The GMAS researchers defined a single common interoperability API, the GMAS API, which supports agent registration, lookup, messaging, launching and mobility.

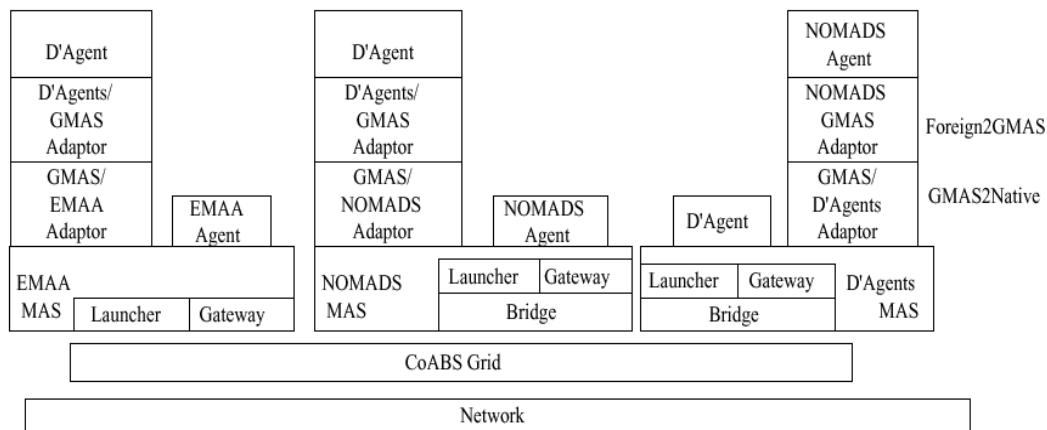


Figure 5.3 GMAS structure

The Figure 5.3 shows the structure of the Grid Mobile Agent System. The current implementation operates over the DARPA CoABS Grid, which connects the different mobile agent systems: EMAA, NOMADS and D'AGENTS. In the diagram we see on each agent system a foreign agent and a native agent. Each layer between the agent and the platform represents an API translation block. The foreign agents can run on top of a non-native agent platform because of two additional layers: first, a layer that translates the foreign MAS API to the GMAS API, next a translation layer for GMAS API to native API.

In addition two components are added to each participating MAS: an Agent Launcher and a Gateway. Any MAS that wishes its agents to operate on others must implement

the Foreign2GMAS translator and the gateway service, while those who are willing to host foreign agents must implement the GMAS2Native translator, the Agent Launcher, and the Gateway.

The distinguishing feature of the GMAS approach is that it does not force a common API to all mobile-agent platforms.

5.3.5 Comparison

For each of the approaches a short comparison is presented. The comparison is purely based on the articles published on the subject since no source code was made available to the public (with exception of the SeMoA system).

Remarkable is that all of the above approaches abstract from the agent's internals. A possible explanation can be found in the fact that our approach involves generative migration and therefore we need to have a compositional agent (assumption 1, see also section 2.3). Other similarities and differences are more special to projects and are discussed next.

Guest Agents

Compared with our approach some differences and similarities are found. With GUEST they define a sort of generic agent that can be supported on other platforms, this holds also for our approach.

Another similarity has to do with the goals of the GUEST project. They explicitly make "providing a uniform view" a goal. Although not explicitly our goal, it is included in our current approach: on every supported platform the IAOF agent has the same view on the platform. Actually, it can be seen in all four approaches that a uniform view of the systems is introduced.

A difference is the following; in our approach the BaseAgent of an agent platform is extended to execute an IAOF agent configuration. In contrast in GUEST, they add a layer between the agents and the agent platform for the support of special GUEST-agents.

Monads

In the Monads project 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. However, rather a distinction would be made between the agent's reasoning process and the wrapper. In contrast with the Monads agent-head our reasoning process consists of components and can be migrated using generative migration whereas they don't mention anything on the compositionality of the head and use JAVA object serialization for migration. Their MAG-service is comparable with our Generative Migration Service.

SeMoA

In SeMoA every non-native agent is wrapped-up to execute in on a SeMoA server. Using a wrapper is similar to our approach. Since they allow JADE agents, our own agents should also be able to execute in the SeMoA environment.

GMAS

GMAS shows more similarities to the GUEST approach than to our approach. In both GMAS and GUEST intermediate interfacing layers are used to allow agents to execute in a non-native environment. However, there are two notable differences. First, GMAS introduces a double-layered intermediate layer where GUEST has just one. Second, GMAS

does not define a special agent or agent interface (API), which is the case in GUEST. Likely, it is possible to program a GMAS agent that can execute on the GMAS2Native layer.

Chapter 6 Discussion

This chapter discusses the work presented in this thesis. First, a brief overview of this thesis is presented. Then, possibilities for future research are described.

6.1 Overview

This thesis focuses on cross-platform generative agent migration. *Agent migration* entails moving agents from one location to another. A new approach to agent migration is called *generative migration*. In generative agent migration an Agent Factory is employed to (re-)generate an agent based on a blueprint. A *blueprint* describes an agent's compositional structure.

To facilitate generative migration two frameworks have been used: ICAMP and IAOF. ICAMP (IIDS Conceptual Agent Modeling Paradigm) is a conceptual framework to model component-based agents. IAOF (IIDS Agent Operational Framework) is an operational framework that executes configurations of compositional JAVA-based agents.

Agent platforms provide environments for agents. They offer various facilities such as life-cycle support (create, delete, suspend, resume), communication, look-up functionality for services and agents (Directory Services), migration, and security. Several agent platforms have been developed over the last decade. *Interoperability* of agent platforms characterizes to which extent two agent systems can work together. Two forms of interoperability are distinguished in this thesis: message interoperability and migration interoperability. The currently well-known standard specification of FIPA and OMG MASIF only partly support these forms of interoperability. Enabling message exchange is troublesome, yet it is possible. Migration of agent between two FIPA-compliant agent platforms, e.g. between JADE and FIPA-OS, is not standardized and not supported.

However, by means of an abstraction from the agent platform agent interface it is possible to enable agent migration. In this thesis generative migration is used to enable exchange of agents between agent platforms. For this purpose, a compositional agent is embedded in a wrapper that is an extension of the `BaseAgent` of the agent platform. The compositional agent is described in a blueprint document, which is written in an independent format that can be exchanged between locations. Additional, state information is used to initialize a newly generated agent on the target location.

The prototype implemented in this project shows that generative migration of agents can be done from FIPA-OS to JADE. No major difficulties were encountered and therefore no major problems are expected for re-use of the design for other platforms.

Other approaches to interoperability and cross-platform migration are based on wrappers, intermediate interface layers, and agent servers. Examples are GMAS, GUEST, MONADS, and SeMoA, which are described and discussed in chapter 5.

6.2 Future Work

In this section subjects for further research are presented. Recommendations are made with respect to security and trust, agent identity, messaging, homogeneous cross-platform migration, self-aware agents, and heterogeneous generative migration.

6.2.1 Security and trust

Security problems are manifold in the field of agent migration. For example, the Generative Migration Service and the Agent Factory need to be trusted services, e.g. the agent does not wish to go to sleep and never wake up. Also, an agent's state may contain information that is strictly private. More research in this area is needed.

6.2.2 Agent Identity

A migrating agent should probably have one identity; whether the identity is made public depends on the application. However, when migrating over different platforms an agent's contact address and name may change. In the current prototype, it may seem to the outside world that the agent assumes a different identity when migrating, since its contact address and name may change.

Another issue is that if the agent name is already present at a destination platform a migrating agent requesting the same name is refused. The agent could handle this situation by changing the requested registration name. A more general approach that needs to be studied is the introduction of (global or system-wide) unique identifiers.

6.2.3 Messaging

Sending messages to migrating agents introduces complicated problems. Sending messages to agents that are on the move brings up the practical problem of where to deliver the message. Solutions are manifold, for example: Installation of a proxy to forward the messages; Using a (global or system-wide) naming-lookup scheme for agents; Home-based approach where an agent retrieves its messages from a fixed location (mailbox). These are aspects that need to be addressed in future work.

6.2.4 Homogeneous cross-platform migration

In the current implementation heterogeneous cross-platform agent migration from FIPA-OS to JADE is demonstrated. In contrast, with the same approach, homogeneous cross-platform migration would be possible, e.g. migration between instances of the same platform. In the case of JADE-to-JADE agent migration no problems are expected. An alternative to generative migration would be to use the JAVA serialisation technology transfer the Agent Image.

With FIPA-OS homogeneous cross-platform migration may be difficult or even impossible to realize since the Generative Migration Service cannot control the agent creation process in the current FIPA-OS version.

Further research and development of the agent platform systems is needed

6.2.5 Self-aware agents

An agent should be aware of its capabilities and be able to reason with so-called self-awareness knowledge (Brazier and Wijngaards, 2002). This knowledge implies knowing about the capabilities needed to perform a task and being aware of the limitations of the current environment. Then, being able to reason and form a plan for successful performance of a task. For example, if an agent moves to the computer network of a hospital. The agent enters a restricted area, where its possibilities for communication with the outside world are limited for obvious security reasons, e.g. it is prohibited to retrieve or publish Internet pages and send messages to other agents outside this platform. An agent needs to be prepared for such a mission and should be aware of the limitations the environment imposes upon the agent. More research in this area is recommended as it could mean a great step forward in the evolution of agents.

6.2.6 Heterogeneous migration

The demonstration of Van Assem (Assem, 2003) shows the possibility of homogeneous generative migration in the AGENTSCAPE OS (AOS) agent platform. This thesis concerns the problem of cross-platform generative migration and demonstrates heterogeneous inter-platform migration (cross-platform migration between different types of agent platforms, e.g. from FIPA-OS to JADE).

The next challenge with respect to the subject of generative migration would be a demonstration of a heterogeneous migration scenario with building blocks in different programming languages.

References

Aart, C.J. van, Marcke, K. van, Pels, R.F. and Smulders, J.L.F.C “International Insurance Traffic with Software Agents”. F. van Harmelen (ed.): ECAI 2002. Proceedings of the 15th European Conference on Artificial Intelligence, IOS Press, Amsterdam, 2002.

Assem, M.F.J. van (2003) “Generative Agent Migration: Exploring Automated Assembly of Agents”, MSc Thesis, Faculty of Sciences, Vrije Universiteit, Amsterdam

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.

Bellifemine, F., Poggi, A., and Rimassa, G. (2000), Developing Multi-agent Systems with JADE. In C. Castelfranchi and Y. Lespérance (editors), *Intelligent Agents VII. Agent Theories, Architectures, and Languages -7th International Workshop, ATAL-2000*, Boston , MAUSA, July 7-9, 2000, Proceedings, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2001.

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

Bellifemine, F., Caire, G., Poggi, A., Rimassa, G., “JADE: A White Paper”. Published Online in Exp Online Magazine, Telecom Italia Lab, Volume 3, nr 3, September 2003.

Brazier, F.M.T., Jonker, C.M., Treur, J. and Wijngaards, N.J.E. (2001), “Compositional Design of a Generic Design Agent”, In: *Design Studies journal*, Vol. 22, Number 5, pp. 439-471

Brazier, F.M.T. Jonker, C.M. Treur, J. ,“Compositional Design and Reuse of a Generic Agent Model”. In: *Applied Artificial Intelligence Journal* Vol: 14 No: 5 , 2000

Brazier, F.M.T., Jonker, C.M. and Treur, J. (2002), “Dynamics and Control in Component-Based Agent Models”, In: *International Journal of Intelligent Systems* (in press)

Brazier, F.M.T. and Wijngaards, N.J.E. (2001) “Automated Servicing of Agents”. In: *Proceedings of the AISB-01 Symposium on Adaptive Agents and Multi-agent systems, at the Agents and Cognition AISB-01 conference*, (ed. Daniel Kudenko, Eduardo Alonso), pp. 54-64, March 2001

Brazier, F.M.T. and Wijngaards, N.J.E. (2002) “Automated (Re-)Design of Software Agents”, In: *Proceedings of the Artificial Intelligence in Design Conference 2002*, (ed. J.S. Gero), pp. 503-520

Brazier, F.M.T., Splunter, S. van, and Wijngaards, N.J.E., 2001, Strategies for integrating multiple viewpoints and levels of detail. (J.S. Gero and K. Hori Ed.) In: Strategic Knowledge and Concept Formation III , Key Centre of Design, Computing and Cognition, University of Sydney. December 2001 , pp. 103-128.

Brazier, F.M.T., Mobach, D.G.A., Overeinder, B.J., Posthumus, E., Splunter, S. van, Steen, M. van and Wijngaards, N.J.E. (2002), AgentScape Demonstration, In: Blockeel, H. and Denecker, M. (editors), *Proceedings of the Fourteenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC2002)*, pp. 513-514

Brazier, F.M.T., Mobach, D.G.A., Overeinder, B.J., Splunter, S. van, Steen, M. van and Wijngaards, N.J.E. (2002), "AgentScape: Middleware, Resource Management, and Services". In: *Proceedings of the 3rd International SANE Conference (SANE 2002)*, pp. 403-404

Brazier, F.M.T., Overeinder, B.J., Steen, M. van, and Wijngaards, N.J.E. (2002a) "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.

Brazier, F.M.T., Overeinder, B.J., Steen, M. van, and Wijngaards, N.J.E. (2002b) "Generative Migration of Agents", In: *Proceedings of the AISB'02 Symposium on Adaptive Agents and Multi-Agent Systems*, (ed. E. Alonso and D. Kudenko and D. Kazakov), pp. 116-119

Broos, R., Dillenseger, B., Dini, P., Hong, T., Leichsenring, A., Leith, M., Malville, E., Nietfeld, M., Sadi, K. and Zell, M. (1998) "Mobile Agent Platform Assessment Report", MIAMI Agent Platforms Contribution of the ACTS Programme project AC338 (MIAMI) to the ACTS D5 Agent Cluster sub-cluster Agent Platforms. Guther, A. and Zell, M. (eds.).

Available at: <http://www.fokus.gmd.de/research/cc/ecco/climate/ap-documents/miami-agplatf.pdf>

Wollrath, A, Waldo, J. (1998) "Client/Server Trail: RMI " Copyright 1995-2003, Sun Microsystems, Inc. Published in "The JAVA Tutorial Continued: The Rest of JDK", Campione, M., Walrath, K., Huml, A., Addison-Wesley Pub Co, 1998, ISBN: 0201485583

<http://java.sun.com/docs/books/tutorial/rmi/index.html>

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.

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

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

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 (accepted paper), <http://www.csai.unipa.it/cossentino/paper/AITA02.pdf>

Cossentino, M. (2002) "Different perspectives in designing multi-agent systems" - AGES '02 workshop at NODe02 - 8-9 October 2002 - Erfurt, Germany (accepted paper)

F. Dignum and M. Greaves, "Issues in Agent Communication: An Introduction", LNAI 1916, pp. 1 – 16, 2000. Springer – Verlag Berlin Heidelberg 2000.

Fipa Inform!, the newsletter of the Foundation for Intelligent Physical Agents, Volume 2, Issue 2, June 2001. Available at <http://www.fipa.org> and <http://sharon.csel.it/projects/jade/>

Fowler, M., Scott, K., "UML Distilled", 2nd Edition, Addison Wesley Longman, 2000

Fuggetta, A., Picco, G. P. and Vigna, G., (1998) "Understanding code mobility." *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995) "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley

Giang, N.T., Tung, D.T., (2002), "Agent Platform Evaluation and Comparison". Pellucid, Slovak Academy Of Sciences, <http://pellucid.ui.sav.sk/TR-2002-06.pdf>

Gray, R. S., Cybenko, G., Kotz, D., Peterson, R. A. and Rus, D. (2002) "D'Agents: Applications and performance of a mobile-agent system" *Software: Practice and Experience*, 32(6):543–573

Arne Grimstrup, Robert S. Gray, David Kotz, Maggie R. Breedy, Marco M. Carvalho, Thomas B. Cowin, Daria A. Chacón, Joyce Barton, Chris Garrett, Martin Hofmann "Toward Interoperability of Mobile-Agent Systems". In: *Mobile Agents 2002*: 106-120

Magnin, L., Pham, T. V., Dury, A., Besson, N. and Thiefaine, 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.

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.

Mäkeläinen, M. (2000) "Agent Mobility support for FIPA-OS". Project report Bachelor of Science in Computing Systems, Nottingham Trent University, Department of Computing.

Milojicic, D., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, B., Kosaka,

K., Lange, D., Ono, K., Oshima, M., Tham, C., Virdhagriswaran, S. and White, J. (1998) "MASIF: The OMG mobile agent system interoperability facility", In: *Proceedings of the 2nd International Workshop on Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, pages 50–67, Springer-Verlag, Berlin.

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.
<http://www.cs.helsinki.fi/research/monads/papers/icdcs2000/gateway.pdf>

Moraitis, P., Petraki, E., Spanoudakis, N., (2002) "Engineering JADE Agents with the Gaia Methodology". Agent Technology and Software Engineering (AgeS) International Workshop of 3rd united GI conference "Object-Oriented Programming for the Net world" (Net.ObjectDays 2002), 2002, Erfurt, Germany. Also in R. Kowalszyk, J. Miller, H. Tianfield, R. Unland (eds), *Lecture Notes in Computer Science (LNCS)*, vol. 2592: "Agent Technologies, Infrastructures, Tools, and Applications for e-Services", Springer-Verlag, 2003, pp 77-91

Nwana, H.S., Ndumu, D., Lee, L. and Collis, J. (1999) "ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems". In *Applied Artificial Intelligence Journal*, Vol 13 (1), 1999, p129-186.

Nwana, H.S. (1996) "Software Agents: an Overview", *Knowledge Engineering Review*, vol. 11(3), pp. 205 - 244.

Peine, H. (2002) "Application and programming experience with the Ara mobile agent system" *Software: Practice and Experience*, 32(6):515–541

Pinsdorf, U. and Roth, V. "Mobile Agent Interoperability Patterns and Practice". In *Proceedings of Ninth IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS 2002)*, Computer Graphics Edition, pages 238-244, University of Lund, Lund, Sweden, April 2002. Institute of Electrical and Electronics Engineers, IEEE Computer Society Press. ISBN 0-7695-1549-5

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

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

Roth, V. Pinsdorf, U., and Binder, W. (2001) "Mobile Agent Interoperability Revisited", 5th IEEE International Conference on Mobile Agents, Keith Marzullo and Amy L. Murphy and Gian Pietro Picco, IEEE Computer Society, p5-8, IEEE Society Press, Atlanta, Georgia, USA.

Scholten, O. (2003) "Modeling Building Blocks for Agent Factories", MSc Thesis, Faculty of Sciences, Vrije Universiteit Amsterdam

Splunter, S. van (2002) "Strategic Automated Agent Design", MSc Thesis, Faculty of Sciences, Vrije Universiteit Amsterdam

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.

Suri, N., Bradshaw, J.M., Breedy, M.R., Groth, P.T., Hill, G.A., Jeffers, R., Mitrovich, T.S. (2000) "An Overview of the NOMADS Mobile Agent System". In Proceedings of ECOOP'2000, Nice, France, 2000
<http://citeseer.nj.nec.com/suri00overview.html>

Tanenbaum, A.S. and Steen, M. van (2002) "Distributed Systems – Principles and Paradigms", Prentice-Hall, New Jersey

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

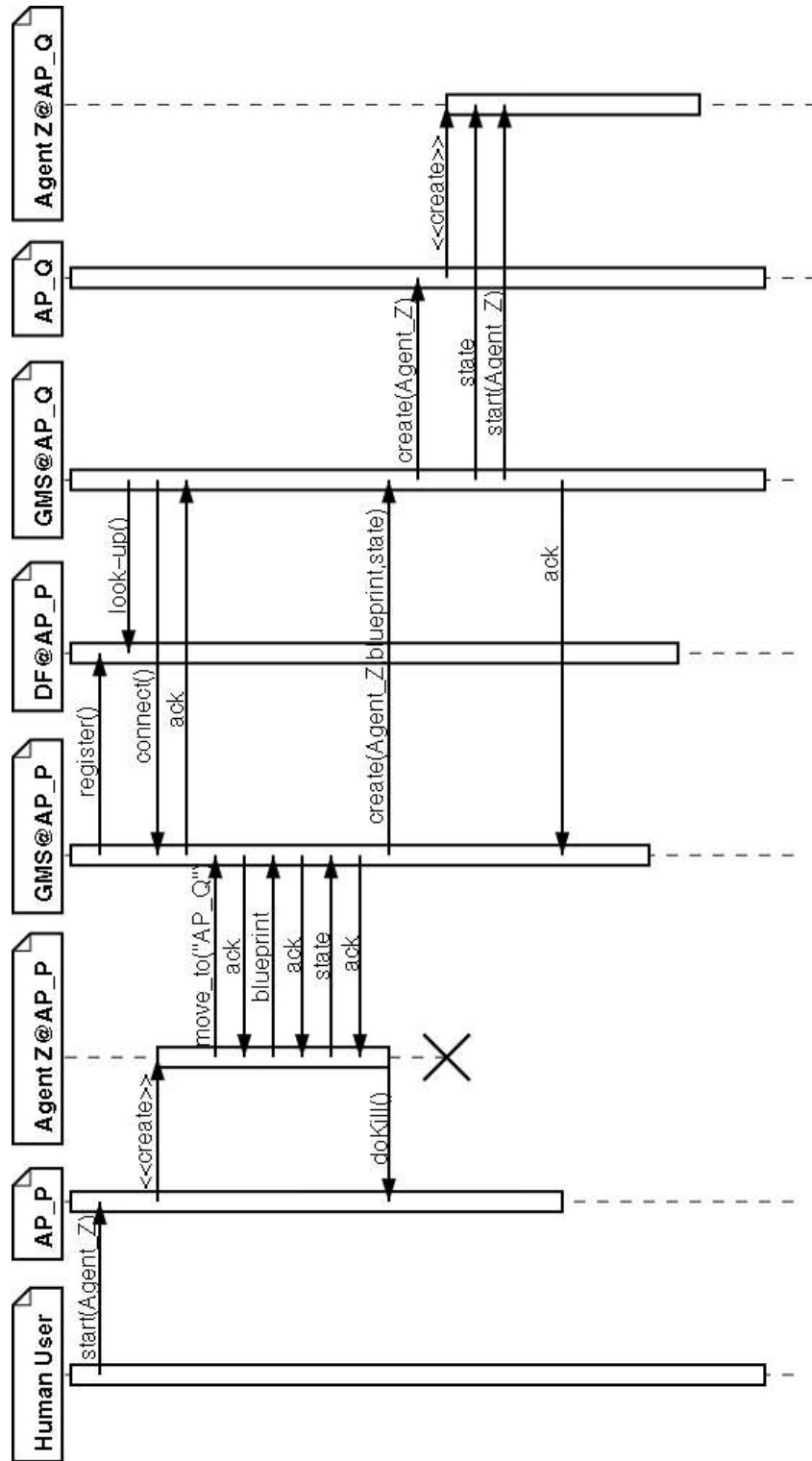
Tripathi, A., Karnik, N., Vora, M., Ahmed, T., Singh, R. (1999) "Mobile Agent Programming in Ajanta", In Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS '99)
<http://www.cs.umn.edu/Ajanta/publications.html>

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.

Wooldridge, M. and Jennings, N.R. (1995), "Intelligent agents: Theory and practice", Knowledge Engineering Review, 10(2), 115-152.

Appendix I

Interaction Diagram 1: agent's state is transferred to GMS and initialized in a new agent.



Interaction Diagram 2: agent's state is not transferred but communicated later to the newly generated agent

