AN APPROACH TO FAILITATING THE TRAINING OF MOBILE AGENT PROGRAMMERS AND ENCOURAGING THE PROGRESSION TO AN AGENT-ORIENTED PARADIGM

M. A. SCHOEMAN

AN APPROACH TO FAILITATING THE TRAINING OF MOBILE AGENT PROGRAMMERS AND ENCOURAGING THE PROGRESSION TO AN AGENT-ORIENTED PARADIGM

by

MARTHA ANNA SCHOEMAN

submitted in part fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

in the subject

COMPUTER SCIENCE

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF E CLOETE

JOINT SUPERVISOR: PROF L M VENTER

DESEMBER 2005

ABSTRACT

Mobile agents hold significant benefits for the rapid expansion of Internet applications and current trends in computing. Despite continued interest, the promised deployment has not taken place, indicating a need for a programming model to introduce novice mobile agent programmers to this environment/paradigm. Accordingly the research question asked was, *"Since novice mobile agent programmers[1] require a paradigm shift to construct successful systems, how can they be equipped to grasp the contextual issues and gain the necessary skills within reasonable time limits?"*

To answer the question, a complete reference providing contextual information and knowledge of mobile agent system development was compiled. Simultaneously novices are introduced to agent orientation. A generic mobile agent system architectural model, incorporating guidelines for programming mobile agents, further provides a framework that can be used to design a mobile agent system. These two structures are presented in a knowledge base that serves as a referencing tool to unlock concepts and knowledge units to novices while developing mobile agent systems.

Key terms: Mobile agents; Mobile agent systems; Agent orientation; Software architectural model, Knowledge base, Novice programmers

[1]Novices include mature, beginner and intermediate programmers who have no experience of mobile agent programming.

ACKNOWLEDGEMENTS

Him who is able to do so much more than we can ever ask for, or even think of

Johan, Chris, Jan-Hendrik and Berto for love, support and sharing our holidays at Kleinbrak with my studies

Lucas and Elsabé for guidance and encouragement

My family, friends and colleagues for prayers and support

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Acronyms used in this dissertation

Acronym	**Definition**	**Page number where first used**
ACL	Agent communication language	2
ADK	Agent Development Kit	15
AI	Artificial Intelligence	6
AML	Agent Modelling Language	68
AOSE	Agent-oriented software engineering	38
API	Application Programming Interface	28
AUML	Agent UML	46
BDI	Belief-Desire-Intention	13
CLAIM	Computational Language for Autonomous, Intelligent and Mobile Agents	13
DAI	Distributed Artificial Intelligence	13
FIPA	Foundation for Intelligent Agents	15
HCI	Human Computer Interaction	8
MASIF	Mobile Agent System Interoperability Facility	15
OMG	Object Management Group	6
OO	Object-oriented/ Object orientation	4
RPC	Remote procedure call	24
SODA	Societies in open and distributed spaces	47
TCP/IP	Transmission Control Protocol/Internet Protocol	75
TUTA	Tshwane University of Technology Agents	101
UML	Unified Modeling Language	43
WWW	World Wide Web	35

CHAPTER 1

1 PROPOSAL

1.1 Introduction

The usefulness and viability of mobile agents have been debated since the mid-nineties, and the debate continues (Gray, 2004; Harrison, Chess & Kershenbaum, 1995; Johansen, 2004; Kotz, Gray & Rus, 2002; Kotz & Gray, 1999; Lange & Oshima, 1999; Milojicic, 1999; Roth, 2004; Roth, Braun & Rossak, 2002; Samaras, 2004; Satoh, 2005; Vigna, 2004; Zaslavsky, 2004). Despite the continued interest demonstrated, the promised deployment has not taken place. The number of publications on mobile agents from 1996 to 1999 and the noticeable decline therein from 1999 to 2001 seem to confirm this. Nevertheless, several researchers have noticed that a steady trickle of articles, workshops and panel discussions related to mobile agent research continues (Roth, 2004; Zaslavsky, 2004). Since the end of 2001 the number of publications has increased again. This is mainly due to the recent interest in the semantic web and the ubiquitous growth of Internet applications. The renewed interest is fuelled by mobile agents' potential to offer a convenient structuring technique in distributed and Internet applications owing to their flexibility, adaptability and capability of migration (Hayes-Roth & Amor, 2003; Johansen, Lauvset & Marzullo, 2002). This is substantiated by the inclusion of mobile agents as one of the topics covered by the recently launched Journal of Mobile Information Systems (Taniar, 2005).

However, many of the problems preventing the widespread adoption of mobile agent technology have not yet been resolved. There are few commercial mobile agent systems able to meet the needs of large complex systems, and even fewer standards[2] (Gray, 2004; Kotz *et al.*, 2002). Researchers agree that the mobile agent model is not generally applied, but they also maintain that it has overcome the first acceptance phases and has been widely accepted as a break-through in obtaining results in some application areas such as systems and network management fields, and the search and filtering of globally available information (Corradi, 2001; Gray, 2004; Johansen, 2004; Zaslavsky, 2004). Some industries, notably the telecommunications and mobile computing industries, are currently actively exploring mobile agents for a range of applications (*2004 IEEE International Conference on Mobile*

[2]Mobile agent standards are addressed in chapter 2.

Data Management (MDM'04), ; Gray, 2004; Manvi & Venkataram, 2004; Satoh, 2004; 2005). Other researchers maintain that agents represent the most important new paradigm for software development since object-orientation (Luck, McBurney & Preis, 2004b; Luck, McBurney, Shehory, Willmott & Community, 2005). They foresee that systems will develop from the current closed multi-agent systems[3], into integrated systems across corporate boundaries that share standard agent communication languages (ACLs), despite interaction protocols still being non-standard. In the medium term much larger systems with large numbers of agents will be developed in which heterogeneous agents, designed by different teams, will be able to participate in multi-agent systems provided their behaviour conforms to a defined set of standards. In the long term (2009 onwards) multi-agent systems are expected to be fully scalable, without arbitrary limits, such as restrictions on agents, users and complexity.

The broad problems in the field, due to independent agent development, are pointed out by Kendall *et al.* (2000):

- lack of an agreed definition - agents developed by different teams differ in capabilities;
- duplication of effort - little or no reuse of agent architectures, designs or components takes place;
- inability to satisfy industrial strength requirements - agents have to integrate with existing software and computer infrastructure while addressing scaling and security concerns;
- incompatibility - if development efforts are not coordinated, agents cannot interact and cooperate with each other in a standardized manner.

These problems are mainly due to lack of a programming model for agent-based applications (Milojicic, 1999). Furthermore, there seems to be general consensus that mobile agents are difficult to design and implement, especially for novices, because of the infrastructure they require and the environment in which they operate (Hayes-Roth & Amor, 2003; Johansen *et al.*, 2002; Luck, Ashri & d'Inverno, 2004a; Vigna, 2004). Therefore it seems as if a model indicating the key features that make mobile agents successful would encourage wider mobile agent deployment (Gray, 2004; Johansen, 2004; Kotz *et al.*, 2002; Kotz & Gray, 1999).

[3]Closed systems are defined as systems designed by one design team for one corporate environment where agents share common goals in a single domain.

Despite the problems pointed out above, a preliminary literature survey done in 2002 and work being done by a number of organizations involved in AgentLink III[4] (*AgentLink III*), appear to confirm the time schedule proposed by Luck *et al*. (2004b; 2005). Current trends in computing, such as pervasiveness, distributed systems interconnected via networks, ambient intelligence and Web Services (Hayes-Roth & Amor, 2003; Luck *et al.*, 2004a; 2005; Wooldridge, 2002), also emphasize the relevance and applicability of mobile agents due to their ability to carry their code with them. Luck *et al.*'s (Luck *et al.*, 2004b; 2005) proposed time schedule, the rapid expansion of Internet applications and wireless computing, as well as the benefits mobile agents hold for mobile platforms (Roth *et al.*, 2002), distributed and Internet applications (Hayes-Roth & Amor, 2003; Johansen *et al.*, 2002), seem to indicate that introducing novice users to mobile agent programming will become a common task in the future.

The high levels of interest in mobile agents, followed by a relatively sharp decline as indicated by the decrease in publications, and the renewed growth in interest mentioned earlier, correspond to the Hype cycle (*Gartner's Hype cycle*). The Hype cycle (*Gartner's Hype cycle*) is used to indicate the maturity of a technology, from initial excitement to disillusionment and eventually, for some, market acceptance. The Hype cycle (*Gartner's Hype cycle*) consists of the following five stages: technology trigger, peak of inflated expectations, trough of disillusionment, slope of enlightenment and plateau of productivity. The sharp decline in interest in mobile agents corresponds to the trough of disillusionment. The renewed interest corresponds to the slope of enlightenment, when more is learnt about the technology, and, as many problems from the trough of disillusionment are solved, standardization takes place. During this period the technology is adopted mainly in the areas that perceive the greatest benefit. During the last stage, the plateau of productivity, the new technology becomes mainstream since benefits and drawbacks are generally recognized. The fact that the telecommunications and mobile computing communities are at present implementing mobile agents, seems to indicate that mobile agent technology is probably currently at the slope of enlightenment. Therefore, a programming model to provide a set of guidelines to build efficient mobile agents will assist mobile agent technology in reaching the plateau of productivity.

[4] AgentLink III represents the European Coordination Action for Agent Based Computing.

1.2 Research problem

The problems described in the previous section, as well as the point on the Hype cycle where mobile agent technology probably resides, point to the need for a programming model to provide a set of guidelines to build efficient mobile agents. The subsequent and recent confirmation that the development of mobile agents is still deemed to be a difficult task (Hayes-Roth & Amor, 2003; Johansen *et al.*, 2002; Luck *et al.*, 2004a; Vigna, 2004) indicates that the problem has not been addressed satisfactorily. Combined with the expected promise mobile agents hold for mobile platforms and Internet computing, this points to the fact that the use of mobile agents will be much more successful if key characteristics can be extracted into a clear, adaptable set of composable software tools with a standard interface, as suggested by various researchers (Gray, 2004; Johansen, 2004; Kotz *et al.*, 2002; Kotz & Gray, 1999).

A long time span (1999 – 2005) has elapsed with little visible progress in published generic mobile agent deployment models, despite the existence of different groups that each understand a conceptual aspect or a group of aspects about mobile agents very well. The possible incompatibility between the general view of a mobile agent and the diverse definitions of agents (Lind, 2001; Luck *et al.*, 2004a) has been raised as one of the potential reasons why industry has not yet adopted agent and mobile technology as widely as could be expected (Samaras, 2004). This seems to indicate that mobile agent implementers need to make a paradigm shift before deployment can be done successfully. Furthermore, both practical experience and the literature (Rothermel & Schwehm, 1998) show that intense theoretical study and practical experience are required for efficient mobile agent programming. Hence, the research question dealt with in this dissertation is: *"Since novice mobile agent programmers*[5] *require a paradigm shift to construct successful systems, how can they be equipped to grasp the contextual issues and gain the necessary skills within reasonable time limits?"* Approximately six months of part-time study or one to three months of fulltime study would be deemed a reasonable time limit for such an endeavour. The actual time required would clearly depend on the proficiency of the novice.

[5]Novices include mature, beginner and intermediate programmers who have no experience of mobile agent programming.

This dissertation aims to answer the question by providing the conceptual background novice mobile agent programmers require and by describing a framework for the construction of mobile agent systems. These two aspects are then combined in a knowledge base that provides a set of guidelines that can be used by novice users, other researchers and practitioners to construct efficient mobile agent systems. A knowledge base is defined as

'*a special kind of database for knowledge management. It is the base for the collection of knowledge. Normally, the knowledge base consists of explicit knowledge of an organization, including troubleshooting, articles, white papers, user manuals and others. A knowledge base should have a carefully designed classification structure, content format and search engine.'* (*Knowledge Base*, 2005).

A knowledge base has been chosen to present the conceptual background and construction framework in order to limit the amount of time spent on becoming *au fait* with the field of mobile agents. This decision is based on the ability of a knowledge base to provide quick and easy access to references and information.

1.3 Proposed solution

The research question will be answered by compiling a knowledge base that can be used to introduce novice users to programming mobile agents, as well as to the milieu in which this is done. This knowledge base will provide this by combining the conceptual background and the framework for the construction of mobile agents. The knowledge base will thus

- explain what mobile agents are;
- elaborate on their characteristics and the environment in which they operate;
- indicate the circumstances under which using mobile agents will be appropriate;
- describe the context for programming mobile agents;
- recommend a framework for constructing a mobile agent system; and
- supply a set of guidelines for programming mobile agents with the focus on programming concepts.

The first four points provide the conceptual background to support the paradigm shift, while the last two points explain how to implement it. The knowledge base aids in accessing this knowledge.

1.4 Strategies for finding solutions

A combination of research techniques is used in this enterprise, namely literature survey, modelling and constructing arguments:

- Literature survey: For the purpose of this study, a technical survey will be conducted of the mobile agent technology.
- Modelling: A model presenting a generic mobile agent system architecture will be constructed.
- Constructing arguments: Arguments regarding the paradigm shift required, the value of both the generic mobile agent system architecture and that of the knowledge base will be constructed.

1.5 Research design

A number of researchers claim that agent orientation is an extension of OO (Lind, 2001; Luck *et al.*, 2004a; Odell, 2002; Parunak, 2000; Travers, 1996; Wooldridge, 2002; Zambonelli & Omicini, 2004). Therefore, the literature review includes *inter alia* identifying the differences between programming agents and mobile agents and OO programming in order to introduce the required paradigm shift. In this regard the literature study will also serve to identify essential requirements for implementing a mobile agent system in the agent orientation paradigm. This is used to model a generic mobile agent system architecture.

Based on the literature review and on the essential elements identified and used to construct the generic mobile agent system architecture, a knowledge base is developed. The essential concepts novice mobile agent programmers need to know to enable them to build commercially viable systems will be extracted from the literature survey and the generic mobile agent system architecture in order to populate the knowledge base. A small number of test subjects will be engaged to determine the suitability of the knowledge base as a tool to assist novice mobile agent programmers.

Arguments will be constructed to contend that the paradigm shift, the generic mobile agent system architecture and the knowledge base will certainly aid novice users in building mobile agents.

1.6 Relevance and significance

The relevance of this research has already become apparent in the previous discussions regarding current trends (Hayes-Roth & Amor, 2003; Luck *et al.*, 2004a; 2005; Wooldridge, 2002), mobile agents' potential for both mobile platforms (Roth *et al.*, 2002) and distributed and Internet applications

(Johansen *et al.*, 2002), and the importance of agents as a relatively new programming paradigm (Luck, McBurney & Preist, 2003; Luck *et al.*, 2005).

Developing agent systems requires specialized skills and knowledge in various areas (*AgentBuilder*, 2004; Badjonski, Ivanovic & Budimac, 2005; Dastani & Gomez-Sanz, 2005). Accordingly, no matter what computing background the novice may have, he or she usually has to assimilate new knowledge in order to implement mobile agents. This is part of the required paradigm shift. Novices with a distributed systems background, for example, typically lack the artificial intelligence (AI) training to incorporate intelligence in their agents, while novices from an AI background, on the other hand, lack the distributed systems experience. Therefore several researchers indicate that agent construction toolkits should be used to build agent systems (Dastani & Gomez-Sanz, 2005; Hayes-Roth & Amor, 2003; Luck *et al.*, 2004b; 2005). However, even when using agent construction toolkits to implement agent systems, mobile agent programmers need to be aware of a substantial number of concepts at various levels, such as at the level of individual agents and the agent system level. Accessible articles and toolkits often focus on a specialization area linked to a specific domain, which may not include all the required concepts when it is used as a starting point and applied to a different area. Novices also frequently have difficulty distinguishing between concepts on the different levels and in different contexts, since there is some overlap, as well as in achieving a broad overview of the field. Furthermore, to program agents requires adapting to a different mindset or paradigm shift, namely agent orientation. This research is therefore both necessary and relevant in order to promote the mobile agent paradigm and agent orientation.

The generic mobile agent system architecture presents a model that makes it possible to include different mobile agent characteristics with maximum reuse of available technologies and architectures. Benefits of this model are that it provides a clear understanding of the architectural issues in mobile agent computing, giving novice researchers and practitioners who enter the field for the first time a foundation for making sensible decisions when researching, designing and developing mobile agents. These benefits apply to researchers and practitioners from all communities involved in researching and programming mobile agents. The model is also significant in that it provides a benchmark for researchers and developers to measure the capabilities of mobile agent systems created by commercially available toolkits.

The research is significant to both industry and academia, as it will facilitate the transfer of agent technology and skills by providing a substantial part of the material needed for an introductory course in programming mobile agents. Such a course directly benefits lecturers who are experienced programmers, but not mobile agent programmers, and who have to take part in supervising mobile agent projects, for example at fourth year level. It is also advantageous to novice mobile agent programmers by providing both a broad overview and a reference tool to refresh concepts quickly while exploring the literature and implementing mobile agent systems. Finally, mature programmers embarking on programming mobile agents for the first time may also benefit significantly from an introductory course in programming mobile agents. Mature programmers can generally not afford to spend a substantial amount of time (in terms of calendar weeks) to acquaint themselves with any new field before being productive.

1.7 Context of research

A number of disciplines, such as programming, teaching programming, software engineering and distributed computing, as well as the various application areas for mobile agents, for example mobility in distributed applications and e-commerce, relate to this research.

The context necessitates an investigation into the difference between programming mobile agents and OO programming as well as recommending a framework that incorporates guidelines to be used for constructing mobile agents. Such an investigation relates to programming and teaching programming. The design and development guidelines incorporated in the generic mobile agent system architecture relate to software engineering, though the focus of the research is not on architectural design. The applicability of mobile agents touches on both distributed computing and the various application areas for mobile agents, such as distributed information retrieval, e-commerce and network management.

1.8 Scope of the study

In the previous section on the context of the research, it is shown that this particular research effort relates to many fields. It would therefore be very easy to be distracted into issues concerning related fields. Educationists, for example, will be very interested in the efficacy of learning through the knowledge base. Similarly, software engineers could be interested in the architecture of the knowledge base and in factors that can optimize its design. The Human Computer Interaction (HCI) community might be interested in the commonality between the efficiency of learning through the knowledge base and incorporating additional resources such as mobile agent toolkits.

This research, however, focuses on using the knowledge base to convey fundamental theoretical concepts, since a better understanding of the proper principles of agent orientation will allow developers to build more robust systems that are easier to maintain and expand (Ashri & Luck, 2002). Since the focus is on providing the conceptual knowledge that novice mobile agent programmers require, and enough detailed work has been covered for the requirements of this dissertation, programming constructs as such have been excluded to a large extent. Though agents can also be compared to software components[6], this research focuses on fundamental concepts for novices and agent orientation is therefore compared to object orientation. Furthermore, since mobile agent programming is seen as a relatively young paradigm, references to circumstances for which using a mobile agent can be recommended will be included, though this will not be the main theme. Also, extensive depth of some issues that developed into research topics in their own right, for example mobile agent security, will be covered, but will not form the main focus.

1.9 Synopsis

In Chapter 2 mobile agents, their characteristics and the environment in which they operate are discussed to explain essential concepts. The context for programming mobile agents is sketched in Chapter 3 by exploring the difference between agent-orientation and object-orientation in order to provide the paradigm shift. Based on key concepts extracted from literature, a model of a generic mobile agent system architecture proposes guidelines for programming mobile agents in chapter 4. In Chapter 5 one possible visualization of the knowledge base is described. The research is concluded in Chapter 6.

[6] A component is a cohesive unit of functionality that can be independently developed, delivered and combined with other components to build a larger unit. A component is typically (though not necessarily) implemented as a collection of classes and has contractually specified interface(s) which define its access points. With a pure OO approach, software is built from a collection of classes, while with a component-based approach software is built from a collection of components (Ambler, 2001)

CHAPTER 2

2 MOBILE AGENTS: BACKGROUND FOR RESEARCH AND PROGRAMMING

2.1 Introduction

This chapter introduces the concept of a mobile agent and sets the background with regard to the research and the programming of mobile agents. To define a mobile agent, in section 2.2 the essential qualities of a software agent are first identified. This is followed by an explanation of how these qualities manifest in a mobile agent. To orientate the reader with regard to the background of programming mobile agents, a brief history of mobile agent research, the various communities involved in mobile agent research and their views of mobile agents are presented. The life cycle of a mobile agent and its salient characteristics are described. Circumstances for which the use of mobile agent applications can be recommended are based on the advantages and disadvantages of mobile agents, combined with their salient characteristics.

In section 2.3 the execution environment for mobile agents is introduced by clarifying some of the terminology used with regard to the mobile agent programming environment. Mobile agent systems as well as the features exhibited by mobile agent systems are only covered in brief, since these aspects receive more attention in Chapter 4 when a generic mobile agent system architecture is presented. The available standards for mobile agent systems are also covered briefly.

Section 2.4 provides a summary of this chapter.

2.2 What is a mobile agent?

2.2.1 What is an agent?

There is no clear definition of exactly what comprises an agent, mainly owing to the large number of areas in which agents are applied. These range from AI, telecommunications, distributed computing, intelligent user interfaces, e-commerce, power system management and air traffic control to information retrieval management, digital libraries, smart databases and many more (Dale, 1997; Grimley & Monroe, 1999; Lingnau, Drobnik & Dömel, 1995; Luck *et al*., 2004a). In general, though, end-users see software agents as programs that assist people and act on their behalf, by allowing people to designate work to them (Grimley & Monroe, 1999; Lange & Oshima, 1998; Lingnau *et al.*, 1995).

Franklin and Graesser (1996) define a software agent as an autonomous system, that, while situated within an environment, is at the same time part of the said environment. The software agent is able to sense the environment and act on it over a period of time, in pursuit of its own agenda in order to effect what the system will sense in the future. This corresponds to a widely accepted notion of agency which regards an agent as an autonomous software system acting in a continuous Perceive-Reason-Act cycle (as illustrated in Figure 2.1) as part of a dynamic and open environment in order to achieve a goal (Lind, 2001; Luck *et al.*, 2004a). The software agent *perceives* by receiving and processing some stimulus from its environment. It *reasons* by combining the newly acquired information with its existing knowledge and goals, and determining possible actions. It *acts* by selecting and executing one of the possible actions. As a result the state of the environment is changed and new perceptions are generated for the next Perceive-Reason-Act cycle.

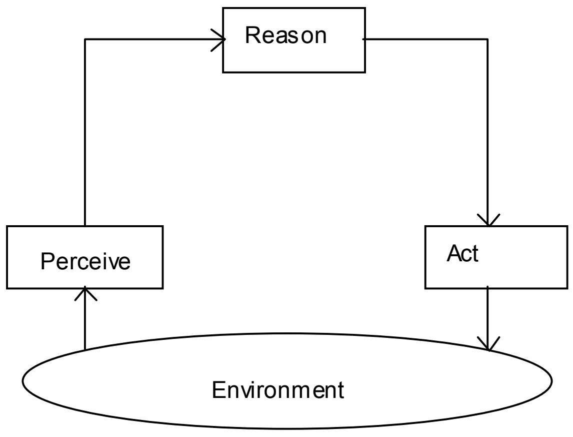


Figure 2.1 The Perceive-Reason-Act cycle illustrated

To succeed in assisting humans, an agent has to meet at least the requirements of *weak* agency (Dale, 1997; Lange & Oshima, 1998; Luck *et al.*, 2004a; Tveit, 2001; Wooldridge & Jennings, 1995), namely

- Autonomy – Once launched, an agent should be able to operate independently without direct programmer or user intervention, make decisions based on information gathered, and control its actions.
- Social ability – An agent must be able to communicate with the local environment, other agents and its user, preferably via an agent-communication language.
- Reactivity – An agent must be able to respond in good time to perceived changes in its environment.
- Pro-activity – An agent must be able to exhibit goal-directed behaviour, based on the state of its environment, to achieve its objectives.

Agents may have additional properties such as the ability to learn, mobility, being rational and/or honest, flexible in the sense that their actions are not scripted, as well as many others (Dale, 1997; Franklin & Graesser, 1996; Luck *et al.*, 2004a; Ndwana & Ndumu, 1996). Adding mentalistic attitudes such as knowledge, belief, intention and obligation to the requirements for weak agency listed above, creates a *strong* agent (Dale, 1997; Luck *et al.*, 2004a; Ndwana & Ndumu, 1996; Wooldridge & Jennings, 1995). Intelligence (reasoning and understanding) can be given to both weak and strong agents, and will determine their behaviour in certain situations as well as their reaction to certain events. The spectrum of agents can therefore vary from weak agents with no specific intelligence but still performing useful tasks, to very strong agents representing nearly the end-goal of AI, a thinking machine (Dale, 1997).

2.2.2 Defining a mobile agent

For the purpose of this dissertation a *mobile agent* is regarded as an active entity that can migrate autonomously through a computer network and resume execution at a remote site to access resources required in order to perform a task on behalf of its user. It is able to observe its environment and to adapt dynamically to changes during its execution. A mobile agent can continue its computations independently even if its user is not connected to the network since both the mobile agent's state and code are transferred with it (Baumann, 2000; Cabri, Leonardi & Zambonelli, 2000; Dale, 1997; Horvat, Cvetkovic, Milutinovic , Kocovic & Kovacevic 2000; Lange & Oshima, 1998; Wooldridge, 2002).

Mobile agents conform to all the criteria for weak agency, namely autonomy, social ability, reactivity and pro-activity. Once launched, the agent decides independently whether it should return to its origin or visit another server (autonomy). During execution, the agent transfers information to the host, and it may receive information from the host (social ability). Once the exchange of information has taken place, the agent will decide, based on this exchange of information (reactivity), whether to terminate, become a resident agent, or repeat the migration process (pro-activity).

This definition does not imply that mobile agents are strong agents, though some mobile agents may contain mentalistic attitudes to enable reasoning such as knowledge, belief, intention and obligation, turning them into strong agents. Agent Factory (Collier, Rooney, O'Donoghue & O'Hare, 2000) and JAM (Huber, 2000), for example, provide the ability to build Belief-Desire-Intention (BDI) mobile agents, while CLAIM (Computational Language for Autonomous, Intelligent and Mobile Agents) can be used to program a multi-agent system consisting of autonomous, intelligent, mobile agents over several platforms (Fallah-Seghrouchni & Suna, 2003a, 2003b).

An individual mobile agent is implemented as a code component and a state component that allows it to continue execution at the destination after migration (Dale, 1997; Harrison *et al.*, 1995; Lange & Oshima, 1999). Since mobile agents are agents, they need an agent execution environment at both the client and host computers. The agent execution environment allows the mobile agent to interact with its host as well as with other agents in order to exchange information or services. It also enables agent mobility, and protects both host and agent security (Harrison *et al.*, 1995; Lingnau *et al.*, 1995). The implementation of a mobile agent as well as its execution environment is discussed in Chapter 4.

2.2.3 Background

2.2.3.1 A brief history

The concept of a mobile agent grew from advances made in distributed systems research. In the distributed systems community mobile agents are mainly intended to improve on remote procedure calls (RPCs) for distributed programming. The distributed artificial intelligence community (DAI) focuses on the ability of a mobile agent to relocate itself between nodes in a network and views a mobile agent as one of a number of different types of agents (Baumann, 2000; Green, Hurst, Nangle, Cunningham, Somers & Evans, 1997; Manvi & Venkataram, 2004; White, 1994; Wong, Paciorek & Moore, 1999). Jim White (1994) coined the term ‘mobile agent’ in 1994 to describe a procedure that travels with its data to a host computer, to be executed at the host, in a white paper describing General Magic’s Telescript technology. The Telescript technology consists of three main components: the Telescript language, an OO and remote programming language specifically designed to program mobile agents and their places (execution environments), an engine or interpreter for the language; and communication protocols to enable engines in different host computers to exchange mobile agents in order to effect an instruction that activates the mobile agent's execution.

Telescript was followed by research systems such as Tacoma in 1995, which offers operating system support for mobile agents written in almost any language, and Agent Tcl in 1996 (the enhanced version is currently known as D'Agents), based on the scripting language Tcl (Baumann, 2000; Tripathi, Ahmed & Karnik, 2001).

Initially a variety of languages was used to develop mobile agent systems. All mobile agent systems are required to be implemented in code that does not require recompilation after migration. Furthermore, they are also required to be executed on a variety of hardware platforms. For this reason, most existing mobile agent platforms are based on either scripting languages or Java. Of these, the platforms based on scripting languages have mostly been developed before Java's general acceptance during the latter half of the nineties (Perdikeas, Chatzipapadopoulos, Venieris & Marino, 1999). While some mobile agents systems such as D'Agents (Gray, 1996; Gray, Cybenko, Kotz, Peterson & Rus, 2002) have been programmed in scripting languages, currently Java is the most commonly used language to program mobile agents (Baumann, 2000; Manvi & Venkataram, 2004; Perdikeas *et al.*, 1999; Pham & Karmouch, 1998; Tripathi *et al.*, 2001; Wooldridge, 2002). Other examples of well-known Java-based applications include Aglets, Voyager and Odyssey. IBM's Tokyo Research Laboratory started development for one of the first industrial and probably the best-known mobile agent systems, Aglets, in 1995, while ObjectSpace's Voyager started in mid-1996, Mitshubishi's Concordia in 1997, and General Magic's Odyssey in 1997. The latter replaced Telescript (Horvat *et al.*, 2000).

At present several general-purpose commercial mobile agent systems exist, such as Voyager (*Voyager's Intelligent Mobile Agent Technology*, 2004-2005), which integrates distributed computing with mobile agent technology, Jumping Beans (*Jumping Beans*) which provides code mobility and the Agent Development Kit (ADK) from Tryllian (*Agent Development Kit*, 2004), a mobile component-based development platform.

Recently (since 2002) Microsoft's .NET framework has also been employed to implement mobile agent systems (Delamaro & Picco, 2002; Mentz, 2004; Vecchiola, Gozzi & Boccalatte, 2003) or integrate mobile agent computation with Web services (Luck *et al.*, 2004a).

During the second half of the nineties, the level of industrial interest in agents initiated standardization efforts in an attempt to enhance the acceptance of agent technology. For mobile agents in particular, two main sets of standards have been compiled: the Mobile Agent System Interoperability Facility (MASIF) from the Object Management Group (OMG) in 1998 (Milojicic, Breugst, Busse, Campbell, Covaci, Friedman, Kosaka, Lange, Ono, Oshima, Tham, Virdhagriswaran & White, 1998) and the Foundation for Intelligent Physical Agents (FIPA) Agent Management Support for Mobility Specification (*FIPA00087 - FIPA 98 Part 11 Version 1.0: Agent Management Support for Mobility Specification*) in 1998. MASIF attempts to provide a minimal interoperable interface for mobile agent systems with specifications for three areas, namely agent management, agent tracking and agent transport, as well as a reference terminology. The FIPA specification set attempts to identify the minimum requirements necessary to grant mobility to agents. The steps needed to integrate with the OMG MASIF standard are also discussed. Unfortunately both groups seem to be inactive at the moment, without having produced complete standards documents (Baumann, 2000; Luck *et al.*, 2004a; Schoeman & Cloete, 2003). Mobile agent standards are covered in more detail in section 2.3.4.

Despite the lack of complete standards documentation, a steady trickle of articles on mobile agent research continues. At the moment some of the focus of mobile agent research appears to move to enhancing mobile agents with intelligence and to the integration of mobile agents in multi-agent and distributed systems, as can be seen from several projects under way in the DAI and distributed agents communities (*Comet Way Java Agent Kernel*, 2005; Ancona, Demergasso & Mascardi, 2005; Collier *et al.*, 2000; Huber, 2000). The amalgamation of the international conferences on mobile agents with other conferences such as the 2004 IEEE International Conference on Mobile Data Management (MDM'04), confirms this trend. Application areas such as mobile computing (*2004 IEEE International Conference on Mobile Data Management (MDM'04)*) and telecommunications (*Mobile Agents for Telecommunications Applications Workshops,* 2002) are also actively investigating and implementing mobile agents. The recently launched Journal of Mobile Information Systems (Taniar, 2005), for example, includes mobile agents as one of the topics covered. In developing a knowledge base for novice mobile agent programmers, this dissertation provides a view of how mobile agents fit into the current programming milieu, contributing to all of these fields. Chapter 3 describes the context for programming mobile agents and introduces the required paradigm shift.

2.2.3.2 Communities currently working in mobile agents

At least two general communities are active in mobile agent research: the DAI and distributed systems communities (Lingnau *et al.*, 1995; Luck *et al.*, 2004a; Mentz, 2004). Some of the more well-known research groups in the DAI community include the Intelligent Agents Group (IAG) consisting of members of the Department of Computer Science at the Trinity College Dublin and Broadcom Ireland (Green *et al.*, 1997), the Department of Computer Science at the University College Dublin with Agent Factory (*Agent Factory*, 2003; Collier *et al.*, 2000), the JADE community (Bellifemine, Caire, Poggi & Rimassa, 2003) and the Distributed and Information Systems Group at the University of Hamburg with Jadex (Braubach, Pokahr & Lamersdorf, 2004a; Braubach & Pokahr, 2002). D'Agents (*D'Agents: Mobile Agents at Dartmouth College*, 2002) from the Center for Mobile Computing at Dartmouth College (*Center for Mobile Computing*, 2005), Ajanta from the Department of Computer Science, University of Minnesota (Tripathi, Karnik, Ahmed, Singh, Prakash, Kakani, Vora & Pathak, 2002), TACOMA network agents (*The TACOMA project [Tromsø And COrnell Moving Agents],* 2001), a joint project between the Departments of Computer Science of the Universities of Tromso, Cornell and California, as well as Voyager® from Recursion Software, Inc. (*Recursion Software - Intelligent Mobile Agent Technology*, 2004) and Tryllian's ADK (*Agent Development Kit*, 2004) originate from research by the distributed systems community.

The DAI community includes the intelligent and multi-agent systems communities, with their main focus on (stationary) agents placed at nodes distributed over the network and cooperating to pursue a common goal. Multi-agent systems consist of a number of autonomous agents that cooperate or compete to achieve a common goal (Manvi & Venkataram, 2004; Wooldridge, 2002; Zambonelli, Jennings, Omicini & Wooldridge, 2001). As mobile agents can be intelligent as well as be part of a multi-agent system, the DAI community considers mobility an orthogonal property of an agent, namely an optional or additional quality (Lange, 1998; Lange & Oshima, 1998; Lingnau *et al.*, 1995; Mentz, 2004; Pham & Karmouch, 1998; Tveit, 2001; Wooldridge & Jennings, 1995).

The *distributed systems* community typically has a background of research in operating systems, distributed systems and object-orientation. They focus on mobility and consider a mobile agent to be code or an object (Lingnau *et al.*, 1995; Mentz, 2004; Pham & Karmouch, 1998).

The research areas of these two communities coincide in the area of *distributed agent computing*, as multi-agent systems are often distributed systems, and distributed systems act as platforms to support multi-agent systems (*About Distributed Agents*, 2004). Multi-agent systems are used to model complex distributed processes. A distributed system consists of a number of autonomous hosts on a network each executing components that interact (via middleware) in order to present an integrated facility to the users of the system. Distributed systems enable multi-agent systems, while multi-agent systems can be seen as a special kind of distributed application. Distributed agent computing is considered to be the area where these two approaches intersect, which also covers mobile code, and therefore mobile agents.

2.2.3.3 Views of mobile agents

Agents have been classified in various categories according to their characteristics or purposes (Dale, 1997; Franklin & Graesser, 1996; Gilbert, Aparacio, Atkinson, Brady, Ciccarino, Grosf, O'Connor, Osisek, Pritko, Spagna & Wilson, 1995; Lind, 2001; Manvi & Venkataram, 2004; Mentz, 2004; Wooldridge & Jennings, 1995). According to Gilbert *et al.* (1995) software agents can be classified in terms of a space defined by three dimensions: intelligence, agency and mobility (see Figure 2.2). In the intelligence dimension agents can be classified according to their capabilities to express preferences, beliefs and emotions, and their ability to fulfil a task by reasoning, planning and learning. The agency dimension denotes the degree of autonomy and authority vested in the agent, which can be measured by the nature of the interaction between the agent and other entities in the system. The mobility dimension denotes the scope of an agent's mobility, which may range from being static or a desktop agent to Internet agents or network agents.

Rothermel and Schwehm (1998) point out that mobile agents can achieve their desired functionality *without* employing capabilities from the intelligence or agency dimensions. Gray (2004) too argues that mobile agents 'really are poorly named' as many mobile agent applications require no intelligence, adaptability or goal-driven behaviour in the moving components, whereas for many agent applications demonstrating intelligent behaviour, stationary code suffices. Most applications therefore require only a subset of agent and mobility behaviours. This endorses the distributed computing community's view of mobile agents, namely that a mobile agent is just an object or program code. Note though that a certain level of autonomy is necessary to allow a mobile agent to migrate without external intervention.

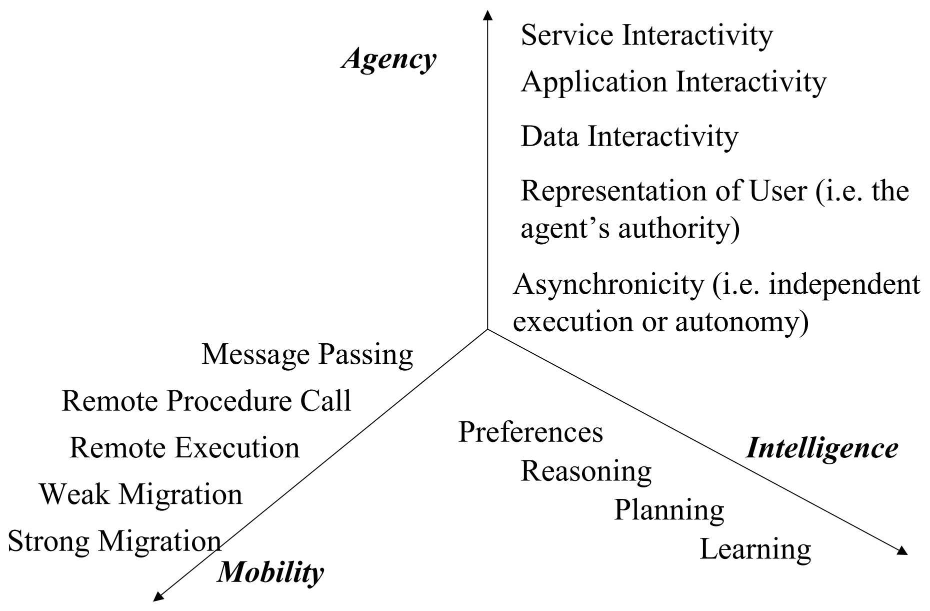


Figure 2.2 Dimensions used to classify software agents according to Gilbert *et al.* (1995)

In contrast, the multi-agent systems and distributed agents communities have several projects going to create BDI mobile agents, that is, strong mobile agents with a level of intelligence that allows them to reason based on goals, beliefs and commitments (*Comet Way Java Agent Kernel*, 2005; Ancona *et al.*,2005; Collier *et al.*, 2000; Huber, 2000). Multi-agent systems *per se* also imply a measure of mobility, as even stationary agents have to be moved at least initially to their nodes, while a multi-agent system can and frequently does, include mobile agents.

Furthermore, the current trend in agent technology appears to be towards multi-agent systems for the development of complex software systems. Cabri, Ferrari & Leonardi (2005) even consider mobility as essential an attribute of agents as the other properties of weak agency (autonomy, social ability, reactivity and pro-activity).

As mentioned previously, the possible incompatibility between the general view of a mobile agent and the diverse definitions of agents has been raised as one of the potential reasons why industry has not yet adopted agent and mobile technology as widely as could be expected (Samaras, 2004). Johansen (2004), for example, claims that a web service down-loaded over the network resembles a single-hop mobile agent. While this invokes some measure of doubt as to exactly what degree of agency is necessary to term a mobile agent an agent, it also serves to emphasize the wide diversity of notions of what an agent and a mobile agent specifically is, as can be seen from the different views held by the distributed computing community, and the DAI and distributed agents communities.

Despite the difference in views and emphasis held by these communities, in essence a mobile agent is an agent albeit one whose level of intelligence, autonomy (demonstrated by its reactivity and pro-activity) and interaction (with other agents, its environment and humans, i.e. its social ability) can vary widely. Since a mobile agent is an agent, and ths more than just an object, novices need to make a paradigm shift in order to program mobile agents.

2.2.4 Mobile agent life cycle

The life cycle of a mobile agent defines the different execution states in which it may be, as well as the events that cause the transition from one state to another (Green *et al.*, 1997).

FIPA (*FIPA00087 - FIPA 98 Part 11 Version 1.0: Agent Management Support for Mobility Specification*) identifies six execution states during the life cycle of a mobile agent: initialised, active, suspended, waiting, migration and disposed. This is illustrated in Figure 2.3. Each circle in the example indicates the state of a particular mobile agent during its life cycle. The mobile agent in this illustration visits two remote hosts. Note that a specific state may occur more than once, and indeed does, during the mobile agent's life cycle. If required, the mobile agent could create clones. Lange and Oshima (1998) also indicate that its owner can control a mobile agent by *retracting* it from a host. The various states in which the mobile agent may be, are numbered in the order in which they occur in this example, and the following explanation holds:

1. A new agent is *created (initialised)* only once by its owner or user on the sending host. At this time it receives a unique ID and initial state, and is prepared for further instructions.
2. The mobile agent is invoked, causing it to *migrate* to a remote host to execute there.
3. Each time the mobile agent arrives at a new host, it is started (activated) with its own thread of execution. In this *active* state the mobile agent can execute independently at the remote host.

3a. The mobile agent creates a *clone* of itself to enable it to pursue its task in parallel on different hosts.

4. On *deactivation (suspension)*, the agent stops processing, and stores its state and intermediate results in such a way that allows it to migrate.

5. During transition/migration the mobile agent is possibly being transferred to another host.
6. On arrival at the next host, the mobile agent is restarted, and continues execution – it is now once again in an active state.
7. When the mobile agent is interrupted to wait for a specific event, it goes into a *waiting* state.
8. Before migrating to the next host, the mobile agent is once again suspended.
9. The mobile agent returns to its owner/user with the results of its computation, by migrating to the sending host. Alternatively, in some mobile agent systems the owner could have *retracted* the mobile agent, which would return the mobile agent to the host from where it was retracted.
10. Once the mobile agent has returned to its owner on the sending host, and the results of its computation has been extracted the owner disposes of it. On *disposal* the agent stops all its operations and frees all resources it has been using, causing its state to be lost permanently so that the mobile agent is destroyed (Braun, Eismann, Erfurth & Rossak, 2001; Horvat *et al.*, 2000; Vogler, Kunkelmann & Moschgath, 1997; Wong, Helmer, Naganathan, Polavarapu, Honovar & Miller, 2001).

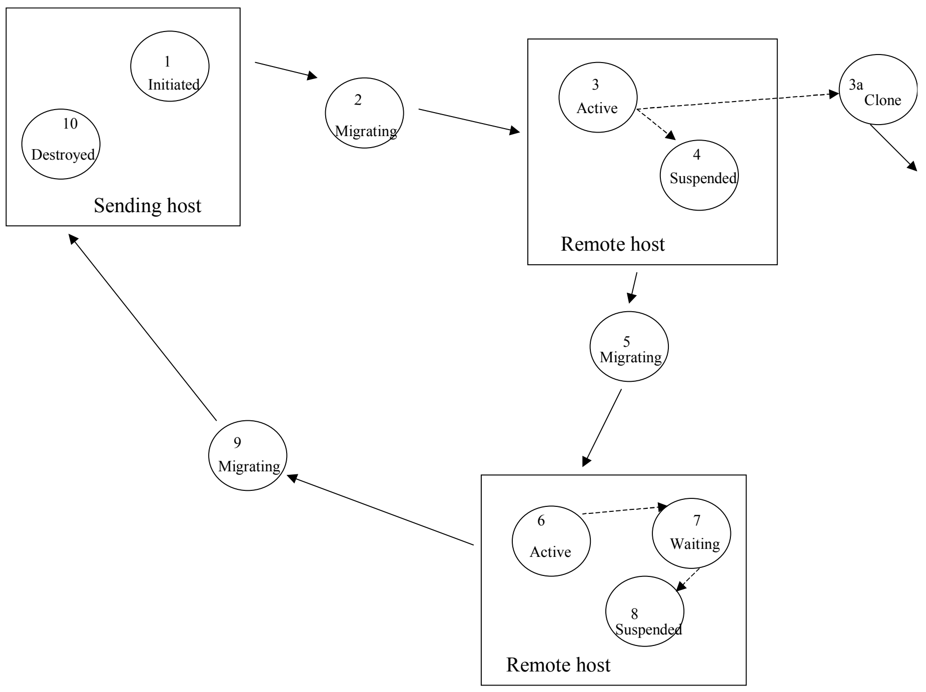


Figure 2.3 The life cycle of a mobile agent

Green *et al.* (1997) identify two life cycle models for mobile agents: the persistent process model (as implemented by Telescript [White, 1994] and AgentTcl [Gray, 1996]); and the task-based model implemented by Aglets (Lange & Oshima, 1998). The persistent process model refers to strong migration in which the mobile agent's running state is frozen before migration and execution continues at the new host from the point where it was frozen at the previous node. The task-based model is used in weak migration where the running state is *not* preserved during migration, but the mobile agent restarts executing from the beginning. During weak migration a mobile agent typically follows an itinerary of tasks, based on a set of conditions. Each task has its own state, but when an agent moves to a new node, the context of the currently executing task is lost (Green *et al.*, 1997). An example where weak migration would be more appropriate than strong miration includes situations where the mobile agent is used to distribute or update code, and therefore has to execute all of its code each time. Using a mobile agent with strong migration would make programming a search for the cheapest ticket easier, though it is not impossible to use a mobile agent with weak migration for such a purpose.

Adapting the description of a mobile agent's life cycle to the task-based model will consist mainly of modifying steps 4 and 6 to:

4. On deactivation, owing to a particular condition that caused the mobile agent to suspend execution, the mobile agent stores its intermediate results, if required (which may not always be neccesary), in such a way that it allows migration to the next host on its itinerary.
6. On arrival at the next host, the mobile agent is restarted, and once again executes its code from the beginning, or from a specific point indicated before migration.

The life cycle of a mobile agent is illustrated in Figure 2.3. During its visit to a host, a mobile agent clearly displays the characteristics of weak agency, as pointed out in 2.2.2. Because of its *autonomy*, the mobile agent can execute independently at a host. During execution, the mobile agent displays its *social ability* by interacting with the host to access resources at the host site and exchange information with the host. Based on the exchange of information with the host, the mobile agent decides on the next action to be taken (*reactivity*), and then executes it (*pro-activity*).

2.2.5 Salient characteristics of a mobile agent

According to Lange and Oshima (1998) mobile agents exhibit the following unique characteristics: object-passing, autonomy, asynchronicity, local interaction, disconnected operation and parallel execution. In Braun *et al.*, (2001) adaptation, communication, cooperation, persistence, and goal-orientedness are added to this list. White (1996; 1994) mentions that a mobile agent also has authority as it represents a real-world individual or organization, and as such can be expected to be loyal to its owner or user. In addition, a mobile agent can clone itself, if necessary, which implies a measure of recursion (Harrison *et al.*, 1995; Horvat *et al.*, 2000; Tripathi *et al.*, 2001). Similar characteristics are pointed out by (Green *et al.*, 1997; Ndwana & Ndumu, 1996; Sunsted, 1998).

As mobile agents match the criteria for weak agency, and mobility is the most fundamental characteristic of mobile agents, their salient characteristics can be described as:

- Mobility – A mobile agent can transfer its code, data and execution state (as well as its itinerary, if used), namely the entire entity representing the mobile agent, to another host in a distributed network of computers, and resume execution.
- Autonomy – A mobile agent can arrive at a destination, execute and move independently without external intervention. This enables a mobile agent to fulfil a given task autonomously and asynchronously.
- Asynchronous – As a mobile agent has its own thread of execution, it can execute asynchronously, i.e it does not require its sending host to suspend execution until the mobile agent returns.
- Local interaction – A mobile agent interacts locally with its host, and can if necessary, dispatch other mobile agents (messenger agents or clones of itself) to enable local interaction at another host.
- Disconnected operation – As a mobile agent runs in its own thread, it can perform its tasks at a host regardless of whether or not the network connection is open. If the network connection is closed, it can wait until the connection is re-opened.
- Parallel execution – Cloning allows more than one mobile agent to execute a task in parallel at different hosts.
- Intelligence – Levels of intelligence can vary, but in general mobile agents' intelligence supports autonomy by allowing a mobile agent to learn from information it obtains in the sense that this information assists it in making decisions such as where to move next. Intelligence therefore sustains reactivity as well as pro-activity.
- Communication – A mobile agent can communicate with other agents, the current agent server, local applications and its owner/user, while agent servers can also communicate with each other, thereby providing the social ability expected of weak agents.

- Cooperation – Cooperation is based on communication between agents and their ability to work together in a structured way, preferably via public protocols, to reach a common goal that supports progress for each agent's local goal.
- Adaptation – Mobile agents can react to dynamically changing (external) situations by choosing an appropriate reaction. This supports the reactivity criteria for weak agency.
- Persistence – The state of a mobile agent is persistent, even if the mobile agent is deactivated and restarted at a later stage, as this is a prerequisite for migration, where agent execution is interrupted and resumed at another host.
- Goal-orientedness – Mobile agents perform tasks on behalf of users, and thus have to achieve certain goals. To be goal-oriented, an agent has to be able to be activated on other agent servers, migrate between agent servers, collect information, adapt or react to changing situations/environments, learn and be persistent. This corresponds to the pro-activity exhibited by weak agents.
- Loyalty – An agent performs computations on behalf of the user or authority that creates it, and is therefore expected to be loyal to its user.
- Recursion – If necessary, a mobile agent can create child agents for subtasks.

The primary trait that distinguishes mobile agents from other forms of mobile code and from other agents is autonomous migration, namely the ability to decide by itself when and where to go, based on the current situation in its environment as well as its goals. In order to achieve this, a certain level of intelligence incorporating both reactivity and pro-activity is required. Receptiveness in order to perceive changes in its environment, adaptation to the perceived changes, and goal-orientedness to guide how to react or act, form the foundation of this intelligence. Furthermore, in the persistent process model migration can only succeed if the state of a mobile agent is preserved during the actual process itself.

Mobile agents can exist in a single agent system or in a multi-agent system. Because of the nature of a multi-agent system, the *social character* of a mobile agent as exhibited by its interaction with the environment and cooperation with other agents and/or mobile agents, will be of more importance than in a single agent system. Since an agent performs tasks on behalf of its user, it is expected to be loyal to the user.

The autonomous, asynchronous character of mobile agents, combined with their mobility, allows disconnected operation and parallel execution. This reduces network traffic, as the mobile agent can execute its task(s) at the remote host(s) and only return to its owner/user with the results of its computations. The type of applications that can benefit from using mobile agents therefore typically involve a fair amount of network traffic and/or need to be able to continue execution autonomously even during intermittent connectivity, namely when only asynchronous interaction is possible.

2.2.6 Circumstances suited to mobile agent applications

The circumstances under which the use of mobile agents are regarded as beneficial can be recognized by reviewing the advantages as well as the disadvantages of mobile agents, and combining these with their salient characteristics.

2.2.6.1 Advantages offered by mobile agents

Researchers (Dale, 1997; Harrison *et al.*, 1995; Lange & Oshima, 1999) cite several advantages to using mobile agents:

- They reduce the network load by processing data at the remote host, instead of transmitting it over the network. Reducing the multiple flows of information in RPC communications to a single mobile agent creates a corresponding reduction in network traffic. This also overcomes the limitations of a client computer with insufficient resources.
- They overcome network latency since they can execute and act locally. In critical real-time systems (e.g. robots in manufacturing processes) that need to respond in real time to changes in their environment, this is especially important.
- They encapsulate protocols. In a distributed system, each host owns the code that implements the protocols needed to exchange outgoing and incoming data properly. Because of the practical difficulties in upgrading protocol code, protocols can become a legacy problem. Mobile agents can transfer proprietary protocols to remote hosts by acting as 'channels' for information exchange, and can also be used by vendors to distribute the client end of a transaction protocol in a device-independent way.
- They execute asynchronously and autonomously, thereby exhibiting persistence. Once a mobile agent has been despatched, it becomes independent of the system that launched it, and will not be affected if the sending node goes down. This offers a distinct advantage to mobile computer users, since they can log off after despatching the agent, and retrieve the result at a later stage.

- They adapt dynamically. As seen in the reactivity attribute defined in weak agency, mobile agents are aware of their environment and respond to changes in it.
- They are naturally heterogeneous. Since network computing is heterogeneous (often both in hardware and software), and mobile agents are usually computer- and transport-layer-independent, in a networked system mobile agents will run on different hardware and different operating systems.
- They are robust and fault-tolerant. Mobile agents' ability to react dynamically to adverse situations and events (pro-activity) allows for robust and fault-tolerant distributed systems. If a host has to shut down, for example, the agents on that machine can be warned and allowed to migrate to another host in the network.
- Mobile agents can easily be customized since they are peer entities and can act as either a client or server, according to their needs.

2.2.6.2 Disadvantages of using mobile agents

The major disadvantage to using mobile agents is seen as the need for highly secure agent execution environments. This includes all security issues such as authentication of both user and server, authorisation, verification and protecting agents during transfer, as well as the resulting performance and functional limitations (Dale, 1997; Harrison *et al.*, 1995; Lange & Oshima, 1999; Roth, 2004; Samaras, 2004; Vigna, 2004).

Other security risks listed include aspects such as protecting the mobile agent's data during transit, limitations in the security provided by Java (the most commonly used implementation platform for mobile agents), guaranteeing agent termination, exactly-once protocols and the administrability of mobile agents (Johansen, 2004; Luck *et al.*, 2004b; Roth, 2004; Vigna, 2004). This dissertation does not focus on security issues, and in this regard the reader is referred to (Farmer, Guttman & Swarup, 1996; Grimley & Monroe, 1999; Vogler *et al.*, 1997; Zapf, Muller & Geihs, 1998) among others.

Lack of universally accepted standards and a pervasive infrastructure are also inhibiting factors (Milojicic, Douglis & Wheeler, 1999; Roth, 2004; Samaras, 2004; Vigna, 2004). Other possible problems put forward for the lack of expected deployment, include lack of a so-called 'killer' application', lack of proper education in mobile agents, and lack of scalability or performance. Lack of scalability and performance refers to the possible negative effect that a large number of mobile agents may have on the network (Harrison *et al.*, 1995; Samaras, 2004; Vigna, 2004). As mentioned earlier, Samaras (2004) also alludes to diverse definitions of what is an agent actually is and its incompatibility with the general view of a mobile agent, as another obstacle to the general adoption of mobile agents.

2.2.6.3 When to use a mobile agent

Their salient characteristics, as well as the advantages they offer, confirm the main reasons cited for using mobile agents:

- Intermittent connectivity, slow networks, lightweight devices and mobile computing, as well as offline processing, namely unreliable and/or limited network capacity.
- Using asynchronous execution, autonomy and persistence for distributed retrieval or dissemination of information; or to execute transactions on large networks covering a large area and a vast amount of information (Dale, 1997; Picco, 2001; Samaras, 2004; Sunsted, 1998).

Two major areas in which mobile agents offer considerable advantages have been identified, namely network management (Corradi, 2001; Feridun & Krause, 2001; Gschwind, Feridun & Pleisch, 1999; Kendall *et al.*, 2000; Milojicic, 1999; Picco, 2001; Samaras, 2004; Shih, 2001; Wong *et al.*, 2001) and information retrieval (Corradi, 2001; Feridun & Krause, 2001; Manvi & Venkataram, 2004; Picco, 2001; Samaras, 2004; Shih, 2001). Other areas where mobile agents are seen as offering potential advantages are disconnected computing, also known as wireless or mobile computing, dynamic deployment of code, 'thin' clients or 'resource-limited' devices, personal assistants (used in e-commerce for example), telecommunications networks services, grid computing, workflow and groupware management, monitoring and notification, and mobile agent-based parallel processing (*Voyager's Intelligent Mobile Agent Technology*, 2004-2005; Horvat *et al.*, 2000; Kendall *et al.*, 2000; Lange, 1998; Luck *et al.*, 2004a; Manvi & Venkataram, 2004; Milojicic, 1999; Samaras, 2004; Shih, 2001).

2.3 The mobile agent execution environment

2.3.1 Terminology

Mobile agents need an environment in which they can execute, migrate and communicate. Various authors use the terms 'platform' (Tahara, Ohsuga & Honiden, 1999), 'framework' (Feridun & Krause, 2001), 'architecture' (Gschwind *et al.*, 1999; Shih, 2001; Tahara *et al.*, 1999; Vogler *et al.*, 1997; Wong *et al.*, 2001) or 'infrastructure' (Aridor & Oshima, 1998; Tripathi *et al.*, 2001) to describe this environment. Although there may not be general consensus about the exact meaning of these terms, they describe more or less the same entity. However, some clarification, as described in Huhns & Singh (1998), follows:

A*gent architectures* view agents as reactive/proactive entities and conceptualize agents as consisting of perception, reasoning and action components.

Agent system architectures consider agents as interacting service provider/consumer entities. Agent system architectures facilitate agent operations and interactions under environmental restrictions and allow them to use available services and facilities.

Agent frameworks are programming toolkits for constructing agents, such as D'Agents (*D'Agents:Mobile Agents at Dartmouth College*, 2002) or Voyager® (*Voyager's Intelligent Mobile Agent Technology*, 2004-2005).

Agent infrastructures provide the rules that agents follow to communicate and understand each other in order to share knowledge. Agent infrastructures deal with:

- Ontologies – These allow agents to agree about the meaning of concepts.
- Communication protocols – They describe languages for agent communication.
- Communication infrastructures – These specify channels for agent communication.
- Interaction protocols – They describe conventions for agent interaction.

A *platform* consists of the computer hardware as well as the basic software that is available, such as the operating system (Cuadron, Groote, Van Hee, Hemerik, Somers & Verhoeff).

A mobile agent system is usually programmed using an agent framework (toolkit). An agent framework implements a specific agent system architecture and is built to run on a particular platform. In the mobile agent system, agents will communicate and cooperate according to the rules provided by the agent infrastructure. An agent itself will typically, though not necessarily, be constructed according to a specific agent architecture.

2.3.2 The mobile agent system

A mobile agent system provides the environment mobile agents need to interact with hosts and other agents, to migrate, and to protect both host and agent security. The basic mobile agent environment consists of agencies and authority application programming interfaces (APIs). An agency may also be regarded as an agent server and the authority API as an agent client.

An agency running on each computer is in charge of all the agents executing on that computer, and will know about the other agencies in its neighbourhood. Its tasks include accepting agents, providing suitable runtime environments, supervising agents' execution, organising agent transfer to other hosts, managing communications among agents as well as between agents and hosts, and doing authentication and access control for all agent operations. Each agent will run in a dedicated runtime environment (provided by the agency), which acts as an interface between the agent and its host. A host can contain more than one agency.

The user will interact with the mobile agent system via an authority API that will allow the user to create and control agents locally or directly at remote hosts (Aridor & Oshima, 1998; Dale, 1997; Farmer *et al.*, 1996; Gray *et al.*, 2002; Harrison *et al.*, 1995; Lingnau *et al.*, 1995; Tripathi *et al.*, 2001; Wong *et al.*, 2001).

Though implementing an agency and an authority API is beyond the scope of this dissertation, a generic mobile agent system architecture is discussed in Chapter 4 in the interests of completeness.

2.3.3 Features exhibited by mobile agent systems

It has been argued that the most important characteristics of a mobile agent are its mobility and autonomy, since, as mentioned previously, many mobile agent applications require no intelligence. To support this, mobile agent systems need to exhibit features to support agent mobility, security, communication, access to local resources, fault-tolerance and interoperability (Tripathi *et al.*, 2001). Each of these aspects will be expanded on in Chapter 4.

2.3.3.1 Agent mobility

Not all of the instructions in a mobile agent have to be executed on the same node or even within the same network. Mobility allows the transfer or migration of a mobile agent to another host, as well as the resumption of execution at the new host. Agent migration can be implemented as weak mobility or strong mobility, as discussed in 2.2.4.

2.3.3.2 Naming services

A global naming scheme and name service are needed to locate resources, specify agent servers for migration and establish inter-agent communication (Tripathi *et al.*, 2001), while globally unique agent names are used for identification, controlling and locating agents (Milojicic *et al.*, 1998).

2.3.3.3 Communication

Mobile agents need to communicate with each other, their owners, their hosts and sometimes, with other non-agent services and resources, in order to fulfil their social ability.

2.3.3.4 Access to local resources

Access control to local resources includes controlling the mobile agent's access to application-defined resources and system level resources such as files, I/O devices and network ports, as well as limiting resource consumption, for example CPU time, disk storage, number of threads, network bandwidth, etc. Various mobile agent systems handle this in different ways. Chapter 4 provides more information.

2.3.3.5 Fault-tolerance/Persistence

Picco (2001) points out that, while improved fault-tolerance is cited as one of the benefits of mobile agents, this is only true if agent migration itself is fault-tolerant with proper mechanisms for local recovery in place.

Various situations, such as breakdown of connections or hosts, destruction of the agent or network errors causing the agent to get lost, can prevent an agent from migrating successfully (Horvat *et al.*, 2000; Vogler *et al.*, 1997). In order to ensure a fault-tolerant system, some form of failure detection and recovery has to be provided to ensure persistence.

2.3.3.6 Interoperability

Interoperability refers to both the interoperability between different mobile agent platforms and integration with existing applications. MASIF and FIPA's standardisation efforts attempt to address these issues, and are reviewed in section 2.3.4.

2.3.3.7 Agent management and control

Agent management and control should allow tracking of an agent in order to control its lifetime and goals, as well as a mechanism for recalling or terminating remote agents.

2.3.3.8 Security

Though security is perceived as one of the main issues in preventing a general acceptance and use of mobile agents (Milojicic, 1999), and a significant amount of research is done in this area, there are also claims that in some situations or environments (e.g. in Intranets) the importance of this issue is overestimated (Kotz *et al.*, 2002; Kotz & Gray, 1999; Milojicic, 1999; Picco, 2001).

Nevertheless, the two most important problems in this area are mutual protection of the host and the mobile agent from each other. The host needs to be protected against attacks by unauthorised or malicious mobile agent code, while an agent needs to be protected from the host that is executing it (Kotz *et al.*, 2002; Picco, 2001; Tripathi *et al.*, 2001). Apart from protecting the agents and hosts from each other, the agent system must also be able to handle many other security considerations, including: control resource consumption, protect agents and the data they collected during transmission, as well as during execution, protect the naming services and provide secure remote communication (Lingnau *et al.*, 1995).

2.3.4 Mobile agent standards

As mentioned previously, two main sets of standards have been compiled for mobile agents: MASIF from the OMG (Milojicic *et al.*, 1998) and the FIPA Agent Management Support for Mobility Specification (*FIPA00087 - FIPA 98 Part 11 Version 1.0: Agent Management Support for Mobility Specification*). Though both groups are currently inactive, and the specifications incomplete, they still offer valuable guidelines for novice mobile agent programmers.

2.3.4.1 The FIPA standard

The *FIPA specifications* (*FIPA:The Foundation for Intelligent Agents*, 2000) represent a collection of standards which are intended to promote the interoperation of heterogeneous agents and the services they represent. These specifications are focused on ACLs, agent services and supporting management ontologies for agent systems in general.

Though FIPA is no longer active in the area of mobile agents, the FIPA Agent Management Support for Mobility Specification (*FIPA00087 - FIPA 98 Part 11 Version 1.0: Agent Management Support for Mobility Specification*) intends to cover the minimum requirements necessary to allow agents to take advantage of mobility. This comprises four main areas:

- Definitions of terms related to mobility such as migration, mobile agents etc.

- The minimum specifications for management actions that a FIPA platform would need to provide in order to support mobile agents
- An ontology specification for elements of an agent profile describing items such as agent data and code in a standard framework
- A mobility life cycle, as described in 2.2.4, and a number of different mobility protocols describing different modes of agent transfer between systems.

FIPA does not define how mobile agents or mobile agent systems operate or are implemented. However, it indicates three types of agent mobility, namely migration, cloning and invocation. It also specifies two mobility protocol abstractions that can both be used for all three types of mobility, namely Simple Mobility Protocols and Full Mobility Protocols. In Simple Mobility Protocols the agent relies on a high level protocol that uses a single action to transfer the agent to a destination host. The agent delegates the mobility operation to the host agent platform, which has to implement the protocol and complete the entire migration process. In Full Mobility Protocols the agent directs the mobility protocol itself without delegating responsibility to the host agent platform. The agent first moves its agent code and state to the destination host, and only transfers its identity and authority once it is assured that the new agent has been created successfully.

2.3.4.2 The MASIF standard

The *OMG MASIF* (Milojicic *et al.*, 1998) is a standard specifically aimed at addressing interfaces between mobile agent systems. It presents a set of definitions and interfaces that provide an interoperable interface for mobile agent systems with the purpose of promoting interoperability between agent systems. MASIF uses two primary interfaces to achieve this purpose, namely the *MAFAgentSystem* and the *MAFFinder* interface. These two interfaces address the four interoperability concerns MASIF identified, namely:

- a standard way of managing agents, including operations such as creation, suspension, resumption and termination;
- a common mobility infrastructure enabling different mobile agent systems to communicate with and visit other agent systems;
- a standardised syntax and semantics for naming services of agents and agent systems; and
- a standardised location syntax for finding agents.

The intention of the MAFAgentSystem is to address the first two concerns, while the MAFFinder interface has to realise the last two.

Although the MASIF standard goes some way towards providing a standard for mobile agent systems, it excludes a number of important architectural components in its standardisation attempts. Its claim to interoperability is not as penetrating as desired, since it only addresses interoperability between agent systems written in Java. MASIF justifies this by assuming that Java is becoming the *de facto* standard, which means that all mobile agent systems will be written in Java soon. MASIF does not address local agent operations such as agent interpretation and execution. It also excludes inter-agent communication from its standardization efforts, and does not address conversion issues as such, as these are too complex (Milojicic *et al.*, 1998). Assuming mobile agent systems to be written in Java, the MASIF standard uses built-in Java constructs, as well as CORBA services, to address security and further expects the communication infrastructure to honour certain security concerns. A brief depiction of mobile agent aspects that are addressed by MASIF follows.

MASIF identifies an *authority* as the person or organisation for whom an agent acts. The term *agent system* denotes the platform on a host where agents are created, interpreted, executed, transferred and terminated. Such an agent system is uniquely identified by its name and address, and each host can contain many agent systems. Within each agent system, there can be one or more *places*. A place is the execution environment, within a specific context, where a mobile agent executes. An authority can own many agent systems, which may not all be of the same type. Agent systems belonging to the same authority are known as a *region*, and this is regarded as a security domain as it creates a trusted environment.

The *MAFAgentSystem* interface addresses mobility as follows: To process a mobile agent's request to move, the source agent system halts the agent’s thread, identifies its state and serialises the agent and its state. The serialised data is then encoded according to the underlying transport protocol and authentication information provided to the server before transferring the agent. On the receiver host side, the receiving agent system verifies that it can support the agent profile (agent system type, language and serialisation method) before authenticating the sender. Thereafter the serialised data is decoded and deserialised, whereupon the agent is instantiated. Finally the agent’s state is restored and execution is resumed.

MASIF's naming and locating services are based upon and use corresponding CORBA services. The *MAFFinder* interface defines mechanisms to register, unregister and locate agents, agent systems and places. It also provides a set of techniques that an authority can use to locate agents, including

- *brute force*, where the agent is found by searching for it in every agent system in the region;
- *logging*, where the agent is followed by its trail, indicated by leaving its next destination at each server it visits;
- *agent registration*, where every agent registers its current location in a database that keeps the latest information about agents' locations; and
- *agent advertisement*, where only places are registered, and an agent's location is registered only when the agent advertises itself.

2.4 Summary

It has become clear that autonomy and mobility are the two most distinctive attributes of a mobile agent. Accordingly, within the context of this dissertation, a mobile agent is considered to be an active entity that can migrate autonomously through a computer network and resume execution at a remote host. Mobile agents satisfy the criteria for weak agency, namely autonomy, social ability, reactivity and pro-activity.

The concept of a mobile agent developed from advances in distributed computing. At present research on mobile agents by the DAI and distributed systems computing communities concur in the research area of distributed agent computing. The DAI community considers mobility to be an orthogonal property of agents, while the distributed systems community focuses on mobility and considers a mobile agent to be code or an object. Nevertheless, a mobile agent remains essentially an agent with varying degrees of intelligence, autonomy and interaction, depending on the application in which it is applied. This is a fundamental part of the required paradigm shift. Limited network capacity and circumstances requiring asynchronous execution, autonomy and persistence are the main criteria for recommending the use of mobile agent applications. Current trends appear to integrate mobile agents with multi-agent systems and to imbue mobile agents with a level of intelligence that allows them to reason, as well as actively to investigate the implementation of mobile agents in areas such as telecommunications and mobile computing. In Chapter 3 the context for programming mobile agents is described and the required paradigm shift is introduced.

CHAPTER 3

3 MOBILE AGENT DEVELOPMENT CONTEXT

3.1 Introduction

As described earlier, the purpose of this research is to develop a programming model to introduce novices to mobile agent programming. This includes a description of the context for developing mobile agents and the circumstances suitable for using mobile agents, as well as the required paradigm shift.

Current trends in computing indicate the milieu in which mobile agents will be applied. These consist of, among others, ubiquity or pervasiveness, distributed systems interconnected via networks, increasing complexity, delegation or automating processes by giving control to computer systems, and human orientation, where computers are increasingly personalized in terms of human characteristics (Hayes-Roth & Amor, 2003; Luck *et al.*, 2004a; Luck *et al.*, 2005; Wooldridge, 2002). Both the Semantic Web[7] and Web Services[8] are also important developments.

Several of the trends mentioned are Internet-based applications. Many of them can be implemented by using software agents. For example: agents provide the autonomy and ability to interact that allows *delegation* to remote computer systems or automating processes. Autonomy and interaction also allow *personalizing* computers in terms of human characteristics. Mobile agents are especially suited to *distributed computing*.

Mobile agents carry their program code, data and execution state information with them, making their know-how available throughout the network. Their autonomous, asynchronous nature, combined with their mobility, enables disconnected operation and parallel execution. Autonomy also allows mobile agents to exhibit flexible (reactive, proactive and social) behaviour, since they can react to changes in

[7] The Semantic Web is envisioned as a universal information space consisting of ontologies that create machine-readable meaning and structure of the content of the World Wide Web (WWW) so that agents rather than humans can use it. The intention is that every user of the WWW would have a personal agent on his/her computer to assist in the goal of the Semantic Web (Gerber, 2005).

[8] A Web Service is a software system identified by a URI (Uniform Resource Identifier) that allows other software systems to discover it (*Web Services*, Available at: http://www.w3.org/TR/ws-arch. [Accessed 15/8/2004]).

their environment. Coupled with the ability to carry their know-how with them, this allows mobile agents to offer significant benefits for Internet-based applications as well as for ubiquitous (pervasive) and mobile computing applications. However, to realise the full potential of mobile agents in the programming environment sketched above, they should be implemented according to the proper principles involved in agent programming.

As OO programming is commonly taught today and many modern languages support OO principles, it can be expected that most programmers probably have an OO background. Agents are generally hailed as an extension, improvement or specialization of the OO paradigm (Dastani, 2004; Flores-Mendez, 1999; Lind, 2001; Luck *et al.*, 2004a; Luck *et al.*, 2003; Tveit, 2001). In order to introduce novice mobile agent programmers to agent programming (or orientation), agent programming (or orientation) is compared to OO in this chapter. This provides insight into the way in which agents diverge from objects and how mobile agents fit into this current computing milieu. Such a discussion works towards the objective of this dissertation, namely to provide insight into the paradigm shift required for programming mobile agents.

Section 3.2 thus investigates terminology and implementation issues in the *agent orientation* paradigm, while section 3.3 covers the OO paradigm in brief.

Section 3.4 considers agent orientation as an extension of OO. This is done by describing the evolution of programming approaches in section 3.4.1, and comparing agents and objects in section 3.4.2. Similarities between agent orientation and OO are pointed out, while the specialization from OO to agent orientation is highlighted.

Section 3.5 examines the conditions recommended for agent orientation and the way in which mobile agents fit into this context. Section 3.6 provides the conclusion.

3.2 The agent orientation paradigm

According to the Merriam-Webster Online Dictionary (*Merriam-Webster Online Dictionary*, 2004) a paradigm is seen as a 'philosophical and theoretical framework of a scientific school or discipline within which theories, laws, and generalizations and the experiments performed in support of them are formulated'. A paradigm is also described as an overall strategy or viewpoint, namely a mindset, for doing things (Ambler, 2001). Within the software domain paradigms represent *mental* models of systems, which influence the approach and techniques used to analyse and develop systems (Cloete & Gerber, 2004).

3.2.1 Agent orientation: terminology

In section 2.3.1 during the discussion of the mobile agent execution environment, some terminology regarding this environment was defined. Here, in order to put agent orientation in the correct perspective, some of the terminology used in the context of agent-orientation is explained. In both cases, this is done in order to define the terminology as closely as possible to the relevant discussion. Furthermore, the terminology used to describe the mobile agent execution environment, is used in the agent programming milieu, and therefore lies on a different level.

The terms 'agent-based' and 'agent-oriented' appear to be used quite loosely and sometimes interchangeably in the literature (see for example [Jennings, 1999a; 2001]). To eliminate any confusion that may exist, the terms 'agent-based' and 'agent-oriented' are clarified.

In *A Dictionary of Computing*, (2004) the term 'based' implies 'use of' or 'using', while 'oriented' conveys the impression of being 'focused on' or 'geared towards something'. In the same vein, but applied to programming, both Ambler (2001) and Sebesta (1999) describe object-based programming languages as programming languages that natively support some, but not all of the properties of an OO language, and thereby allow the use of objects. OO programming languages in contrast are generally seen as supporting the OO concepts of inheritance, classes, objects, polymorphism and message passing. Following this analogy, *agent-based systems* can be seen as merely using agents, while *agent-oriented systems* are viewed as having the entire development process from systems analysis to implementation steeped in and founded on agent concepts such as autonomy, interaction, intelligence, etc.

Agent-based programming then entails creating a software system that uses a set of collaborating agents to achieve a task(s). On the other hand, *agent-oriented programming* (AOP) is defined as programming the desired behaviour of individual agents *directly* using mental attitudes, in an agent programming language with semantics based on some theory of agency (Luck *et al.*, 2004a; Shoham, 1993; Wooldridge, 2002). This concept is expanded on in the next section.

Judging by the number of articles referring to 'agent-oriented software engineering', this term seems to be well established. This is confirmed by the recent launch of the new International Journal on Agent-Oriented Software Engineering (IJAOSE) (2005). A*gent-oriented software engineering* (AOSE) approaches focus directly on the properties of agent-based and agent-oriented systems and try to define a methodology to cope with all aspects of agents (Bauer & Muller, 2004; Tveit, 2001).

Though agent-based computing is used as an encompassing term to include AOSE and agent programming, 'agent orientation' (similar to 'object orientation' in comparison to 'object-oriented') appears to be a wider and in general a more encompassing term than both 'agent-based' and 'agent-oriented'. The term *agent orientation* is therefore used in this research to include the concepts of both AOSE, and agent programming. Agent programming is usually a consequence of AOSE and in turn includes both agent-based programming (often implemented by using OO techniques and methodologies) and agent-oriented programming (frequently implemented in explicit agent programming languages), as depicted below:

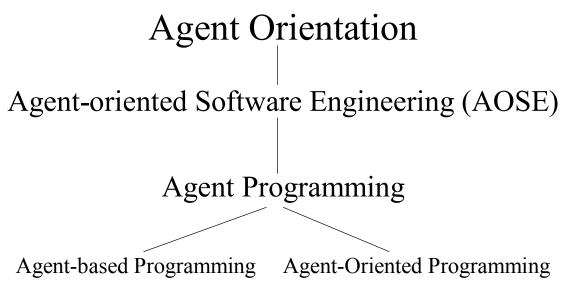

Agent Orientation
Agent-oriented Software Engineering (AOSE)
Agent Programming
Agent-based Programming
Agent-Oriented Programming

Figure 3.1 Agent Orientation

There are essentially two schools of thought in AOSE (Henderson-Sellers & Gorton, 2002). One point of view is that agent-oriented methodologies should be developed independently of OO methodologies. The other believes that existing OO methodologies should be extended to support agent concepts. Methodologies based on the first approach might lead to agent-oriented programming. The latter leans towards agent-based programming, since an OO methodology uses classes, class features and a specific set of relationships based on the object-server model, and were not originally intended to support agent orientation. Similarly, agent-oriented methodologies can probably more readily be used to implement strong agents, than extensions of OO methodologies.

To conclude this section, attention is drawn to the distinction between single-agent and multi-agent systems. In a single-agent system, an agent performs a task on behalf of a user or some process, and consists of a computer program providing assistance to a user dealing with a particular application. During this process, the agent may communicate with a user and local or remote system resources, as well as other agents. Examples include personal assistants, meeting schedulers, mail managers, search agents or news filtering agents. As mentioned earlier, multi-agent systems consist of a number of autonomous agents working towards a common goal. Each agent typically has to achieve a specific task or goal in order to contribute to the agent system's collective goal. This may require the agents to interact with one another, as well as with users and system resources. A single agent may be a mobile agent, and multi-agent systems may consist of mobile agents, stationary agents or a combination of both types (Luck *et al.*, 2004b; Manvi & Venkataram, 2004).

3.2.2 Agent orientation as a paradigm

As explained above, agent orientation includes AOSE as well as agent programming. The philosophical and theoretical framework, or *mindset* of agent orientation as described here, therefore refers to analyzing, designing and implementing complex software systems as a collection of interacting, autonomous agents. These agents exhibit proactive and intelligent behaviour, and interact with one another through high-level protocols and languages (Burmeister, 1996; Jennings, 1999a; Luck *et al.*, 2004a; Zambonelli *et al.*, 2001).

In an agent-oriented software-engineering approach a problem will be decomposed into multiple autonomous agents, embedded in an environment, that can act and interact in flexible ways to achieve their goals (Weyns, Helleboogh, Steegmans, Wolf, Mertens, Boucke & Holvoet, 2004). Three key *abstractions* are employed in the agent-oriented paradigm: agents, interaction and organization (Burmeister, 1996).

Aa already suggested in the previous chapter, an *agent* is autonomous, continuously perceives its environment, reacts to the perceived information and interacts proactively with its environment as well as with other agents in order to achieve its goals. Agents usually act on behalf of individuals/companies or as part of some wider problem-solving effort. *Interaction* between agents therefore typically occurs in the presence of some organizational context, namely in the context of some relationship between the agents concerned, which in turn affects the behaviour of the agents. These relationships have to be represented explicitly in the system. This is done by means of protocols that enable *organizational groupings* to be formed and disbanded, mechanisms that ensure groupings act together in a coherent fashion, and structures that characterize the macro-behaviour of collectives.

In an agent-oriented view of the world, most problems require or involve multiple agents to represent the decentralized nature of the problem, the multiple loci of control, the multiple perspectives or competing interests. Agents in such a multi-agent system need to interact with one another in order to achieve their individual goals or to manage the complexities of being situated in a common environment. There are two points that qualitatively differentiate agent interactions from those that occur in other software engineering paradigms:

- Agent-oriented interactions occur through a high-level ACL so that interactions are conducted at the knowledge level, in terms of which goals should be followed at what time, and by whom. Method invocation or function calls in contrast operate at a purely syntactic level.
- As agents are flexible problem solvers, operating in an environment in which they have only partial control and observability, interactions need to be handled in a similarly flexible manner. Agents therefore need the computational apparatus to make context-dependent decisions about the nature and scope of their interactions and to initiate actions that were not foreseen at the time they were designed (Jennings, 1999a; 1999b; 2001).

Decomposing the system into autonomous *agents* introduces a tool for conceptual grouping that comes with the well-defined bounds of an agent, since all components that compose the control-flow of a particular agent are grouped as an entity. This makes it easier to identify larger units in the system that belong together semantically (Lind, 2001).

As a result agents can be grouped or *organized* in order to be managed as a unit and to create various hierarchies in the system. The *organization* of agents endows a multi-agent system with more than the knowledge, competence and abilities provided by all of its individual agents. Frequently a multi-agent system as a whole, therefore, achieves more than the sum of its component agents' goals. This is termed 'emergence' (Jennings, 1999a; Zambonelli *et al.*, 2001). For example, in a system composed of multiple autonomous agents, each looking for computational resources to utilize, a global load balancing of the activities can be achieved, without any one agent specifically attempting to manage the load balancing. Consequently the organizational structures, or societies of agents, should be analyzed, designed and implemented as first-class entities in the system (Zambonelli *et al.*, 2001).

Huhns (2004) point out that there is a difference between agent-oriented programming and programming multi-agent systems. In agent-oriented programming the goal is to provide a societal view of computation in which multiple agents interact with one another in order to realise their goals. Agents consist of mental states such as beliefs, desires and intentions, while a computation consists of the agents informing, requesting, offering, accepting, rejecting, competing and assisting one another. Various constraints are placed on the mental state and methods of the agents (Shoham, 1993). An agent-oriented program will consist of a set of transition rules that determine how an agent in a given mental state will react to an input, possibly a message or set of messages from other agents, by specifying its new mental state and any outputs (Luck *et al.*, 2004a). Reacting to input and specifying a new mental state, with corresponding outputs, can be seen as (at least) adapting to the environment, and possibly learning from experience.

The abstractions provided by agents are therefore founded on animate behaviour (autonomy and active functionality) as well as psychological concepts such as beliefs, desires and intentions. The combination of activity and autonomy allows an agent to protect its internal state against unwarranted changes and refute unwanted attempts to use its functionality, as well as to play an active role in its environment. *Agent-oriented programming* focuses on implementing the mentalistic states of the *individual* agent (beliefs, desires and intentions) to allow it to act intelligently in its environment. The resultant systems tend to consist of complex entities (strong agents) that use *simple* interactions to deal with a complex environment, as there is little effort to model the environment and incorporate that into the agent (Huhns, 2004).

Multi-agent systems on the other hand are based on social abstractions taken from sociology, economy and law, namely commitments and obligations (Huhns, 2004). In a multi-agent system, agents are regarded as populating a *society* whose global activities proceed according to specific social laws and conventions. Agents living in a multi-agent society therefore have to obey some global laws when interacting with other agents (Zambonelli *et al.*, 2001). Accordingly multi-agent systems focus on cooperative interactions, both between the active entities (agents) and with the passive entities (objects) in their environment, in order to realize the global laws. As a result multi-agent system development models the environment, mutual beliefs, joint intentions and common goals. Commitments are used to encode relationships between the agents, while obligations, namely commitments to follow required policies, enable an agent to reason about the relationship between its responsibilities and its decision-making constraints and goals (Huhns, 2004).

A multi-agent system is therefore a deliberate planned outcome resulting from focusing on the organization and coordination of the agents during the development process, over and above the design and implementation of the agent architectures and their environment.

Figure 3.2 provides an agent-oriented view of a system illustrating some concepts in agent orientation. The system represents a scaled-down version of an examination situation in a school involving a *society* of teachers, learners and a supervisor. *Agents* represent each of the teachers, learners and the supervisor. The school itself is the *environment* in which the agents operate. The teachers (T1 and T2) are *organized* in a group. Each teacher has an *individual goal*, namely to mark the examination scripts for the subject he/she is offering. Each teacher also works towards a *global goal*, namely to determine a final mark for each learner. The final mark consists of the average of the marks for all the subjects a learner is taking. Teachers *interact* with students, for example to hand out marked scripts. They also interact with each other in order to determine the final mark for a learner. The manner in which they interact with each other, differ from the manner in which they interact with a learner, and thus use different *interaction protocols*.

Learners (L1 to L5) are organized in groups according to the subjects they are taking. Similar to the group of teacher, individual goals, common goals, and different interaction protocols, etc. can be identified for the learners as well as for the supervisor S, who supervises during the examination.

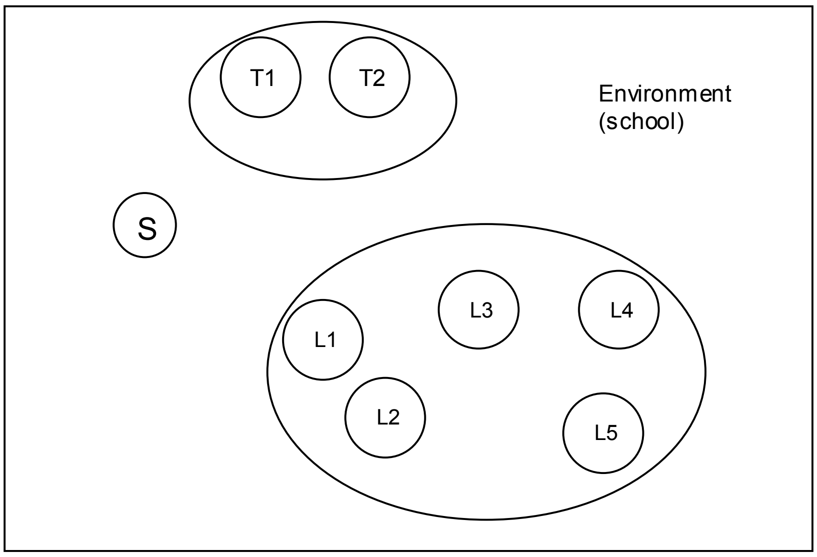


Figure 3.2 The examination situation in a school represented by a society of agents

Since agent orientation is a relatively new software engineering paradigm, a variety of methodologies and techniques to design agent-based and agent-oriented systems has been developed, and no standard methodology has, as yet, been adopted. Various methodologies based on, among others, knowledge-level approaches, agent-oriented approaches and methodological extensions to OO approaches, as well as using patterns and components to develop agents, have been proposed.

3.2.3 Implementing the agent paradigm

Methodologies for software development typically consist of a modelling language and a software development process. The modeling language, such as Unified Modelling Language (UML), is used to represent models of the system visually. The software development process defines the development activities, interrelationships between activities and the manner in which activities are performed. Each activity produces artifacts such as specification documents, designs, models, code, test cases, manuals etc. In the software development process four phases are generally identified: requirements analysis, design, implementation and testing. These phases can be executed sequentially (the waterfall method) or iterations can be made over (parts of) the phases (Ambler, 2001; Bauer & Muller, 2004; Weyns *et al.*, 2004). This dissertation focuses on providing guidelines for novice mobile agent programmers. However, the implementation or programming of mobile agents is based on the analysis and design of agent systems, and therefore a brief look at these phases are necessary to place agent orientation, and specifically mobile agents, in the context of current programming paradigms. Since Chapter 4 considers implementing a mobile agent system, only the analysis and design phases will be discussed here. Section 3.2.4 considers the agent-oriented programming landscape.

3.2.3.1 Requirements Analysis Phase

During the requirements *analysis* phase, the analyst investigates the problem domain and the various requirements by focusing on *what* the problem is and documenting it, without indicating how to resolve it. Both functional requirements and non-functional requirements have to be identified. Functional requirements state what the system must do, namely its capabilities. These are typically captured in use cases. Non-functional requirements refer to constraints on the system such as performance, reliability or security. A visual representation of all relevant concepts or real-world entities in the system is also captured in a domain model (Ambler, 2001; Weyns *et al.*, 2004).

Regardless of which methodology, agent theory, agent architecture or agent language is used to implement a software agent system, at a conceptual level the analysis process should describe both the *agent model(s)*, and *group/society models* for the system to indicate *what* the agent system has to do (but not how it should be done). Agent models state the characteristics of each agent, including its skills (sensors and effectors), reasoning capabilities and tasks. Group/society models convey the relationships and interactions between the agents, as well as their global responsibilities (Iglesias, Garijo & González, 2000; Zambonelli *et al.*, 2001).

Similar to traditional methodologies, agent-oriented analysis starts with the definition of a system's requirements and goals. The application domain is studied and modelled, computational resources available and technological constraints listed, fundamental application goals and targets identified, etc. As agents have goals to pursue proactively and autonomously, agent-oriented analysis should also identify the responsibilities of an agent. These responsibilities consist of the activities it has to carry out in performing one or more tasks. In all probability this involves specific permissions to access and influence the environment as well as interactions with other agents in the application (Zambonelli *et al.*, 2001).

Since a multi-agent system forms a society of agents, the analysis phase should identify the agents' *individual* responsibilities (tasks), as well as their global responsibilities in the multi-agent system as a whole, namely its *social* tasks.

Individual tasks are associated with one specific competency in the system. Such tasks typically involve accessing and influencing a specific section of the environment to perform a simple task. One or more tasks are assigned to each agent in the system, which have to accept responsibility for it. Multiple agents in a system may be assigned the same task. This corresponds to assigning each agent a specific *role* in the organization or society (Zambonelli *et al.*, 2001). The role of an agent is regarded as the functional or social part that an agent embedded in a multi-agent system environment enacts to assist in a (joint) process such as problem solving, planning or learning (Lind, 2002a). A role definition therefore describes the agent's capabilities, the data that is needed to produce the desired results and the requests that may activate a particular service (Lind, 2001).

Social tasks represent the global responsibilities of the agent system. In order to perform these tasks, several, possibly heterogeneous, competencies will usually have to be combined. This creates the need for global social laws that have to be respected and/or enforced by the society of agents in order to function properly and achieve the global task (Zambonelli *et al.*, 2001).

3.2.3.2 Design phase

In the *design* phase the designer creates a solution to the problem based on the outcome of the requirements analysis phase by specifying *how* to solve the problem. Two design levels are commonly recognized: architectural design and detailed design. During *architectural design*, an architecture is devised to satisfy both functional and non-functional requirements. This will denote the structure(s) in the system, consisting of software elements, the externally visible properties of the elements and the relationships between them. Software elements, such as a process, module or component form the building blocks for structures. The externally visible properties of the elements indicate what other elements will know about a specific element such as services provided, performance characteristics and shared resource usage. The relationships between elements indicate how they are connected, namely how an element uses, is used by, relates to, or interacts with other elements. During *detailed design* detailed descriptions and diagrams of the inner workings of all elements in the architecture are created to such a level that the system can actually be implemented (Weyns *et al.*, 2004).

During the *design* of the *agent system*, responsibilities, tasks and interaction protocols are mapped onto agents, high-level interactions and organizations. The competence required by the task(s) assigned to an agent, determine the inner structure of the agent, namely its capabilities and the way it represents the environment. The information required and provided, as well as the actions to be performed to perform the task, in turn determines the interaction protocol for the agent. The agent will be designed to represent the portion of the environment with which it is concerned, to infer new information from this knowledge, and to act on the environment somehow. Its interaction protocols will be defined to allow the agent to contact other agents, obtain the information it needs from them, transfer newly construed information to them and coordinate its activities with other agents (Zambonelli *et al.*, 2001).

Though the design process in the available methodologies may produce a variety of different artifacts, it should provide at least the following in a format detailed enough to allow implementation (or design using traditional OO techniques):

- An internal agent structure or architecture to describe the internal structure of an agent such as its goals, beliefs and plans.
- A social architecture to indicate the social organization of agents or groups of agents, and to describe their behaviour.

- A relationships model to indicate inter-agent dependencies, cooperation and commitments between agents (Bauer & Muller, 2004; Burmeister, 1996; Ellamari & Lalonde, 1999; Sudeikat, Braubach, Pokahr& Lamersdorf 2004, Braubach *et al*., 2004a).

Other design outputs suggested include among others artifacts to indicate (Bauer & Muller, 2004; Ellamari & Lalonde, 1999; Sudeikat *et al.*, 2004)

- the services/capabilities of an agent in order to define the ability of the agent to solve problems autonomously;
- a communication model to allow agents to communicate and cooperate;
- an acquaintance model to provide an agent with models of other agents' beliefs, capabilities and intentions; and
- a deployment model to express the allocation of agents to places in the environment, migration and dynamic creation of agents.

As a substantial number of methodologies and modelling languages are currently used to design agent-based systems, the artifacts produced during the design process vary according to the methodology. These may include among others, UML or Agent UML (AUML) descriptions, use cases, sequence and state diagrams, Class Responsibility Collaborator (CRC) cards, agent class diagrams, protocol diagrams, organization diagrams etc. For a discussion on various methodologies and modelling languages see Bauer and Muller (2004), Iglesias *et al.* (2000), Tveit (2001) and Zambonelli *et al.* (2001).

3.2.3.3 Design approaches

In order to present a comprehensive view of the agent programming environment, some issues in this area are discussed briefly, and a number of design approaches are described.

One of the issues in agent orientation is the gap between agent-oriented design and implementation (Amor, Fuentes & Vallecillo, 2004; Dastani & Gomez-Sanz, 2005; Sudeikat *et al.*, 2004). Currently there is no single unified and unique general-purpose agent-oriented methodology. Some methodologies such as PASSI (process for agent societies specification and implementation) (Cossentino, Burrafato, Lombardo & Sabatucci, 2002), support the complete software development life cycle from problem description to code generation on the agent platform supported by the specific methodology. Most methodologies, however, facilitate only analysis and design. Gaia (Wooldridge, Jennings & Kinny, 2000) and SODA (societies in open and distributed spaces) (Omicini, 2001) are examples of methodologies supporting only analysis and design. In these cases the implementation is done on an existing or available agent platform, which does not necessarily match the methodology used. Developers consequently have to, for each system, define and implement mappings and transformations from the design produced by the agent-oriented methodology, to the APIs provided by the agent platform (Amor *et al.*, 2004).

The diversity of both the domains for agent technology and the underlying agent and multi-agent architectures, implies that a single AOSE methodology may not be suitable for all purposes. Accordingly, which agent-oriented methodology to use, is to some extent dependent on the platform on which the design will be implemented, since different implementations of agent architectures need different levels of expressiveness (Bauer & Muller, 2004; Sudeikat *et al.*, 2004). While this is not necessarily perceived as a problem (*Report on Workshop 14 at OOPSLA2002: Agent-Oriented Methodologies*, 2002; Henderson-Sellers & Gorton, 2002), various approaches attempt to bridge the gap between agent-oriented design and implementation. For more information on these efforts, see (*FIPA Methodology Technical Committee*; *OMG Agent Working Group*; *Report on Workshop 14 at OOPSLA2002: Agent-Oriented Methodologies*; Amor *et al.*, 2004; Ashri & Luck, 2002; Bauer & Muller, 2004; Fortino, Russo & Zimeo, 2004; Henderson-Sellers & Gorton, 2002; Jo, 2001; Sudeikat *et al.*, 2004).

In order not to detract from the focus of this research, this problem will not be discussed further. Only a brief survey of agent-oriented design approaches is hence provided in this section, followed by their implementation in the next section.

Three main roots for the development of agent-oriented methodologies and modelling languages can be identified: extensions of both knowledge engineering and OO approaches, and agent-oriented approaches (Bauer & Muller, 2004; Iglesias *et al.*, 2000; Zambonelli *et al.*, 2001).

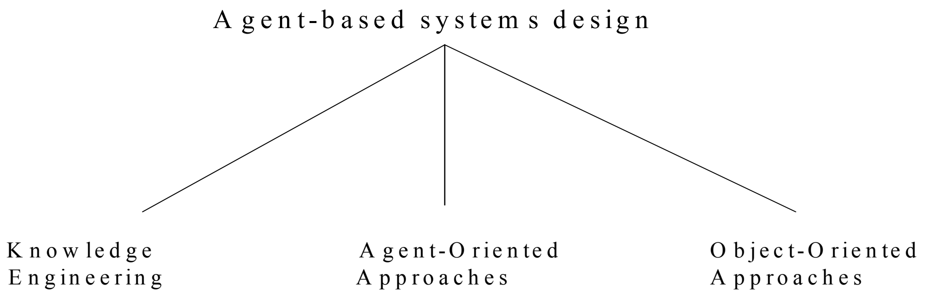

Agent-based systems design
Knowledge Engineering
Agent-Oriented Approaches
Object-Oriented Approaches

Figure 3.3 Roots for the development of agent-oriented methodologies

Early approaches supporting the software engineering of agent-based systems were based on knowledge engineering. These approaches now once again receive attention in Semantic Web and ontology technologies. Examples include CommonKADS (Schreiber, Wielinga, Akkermans & Van de Velde, 1994), CoMoMAS (Glaser, 1996) and MAS-CommonKADS (Iglesias, Garijo, Gonzalez & Velasco, 1997). Since agents have cognitive characteristics, knowledge engineering principles provide the ability to represent entities and relationships relevant for a specific domain from a knowledge-level perspective by providing formal and compositional modelling languages for the verification of system structure and function. However, these approaches cannot support specific agent-related constructs such as the distributed and social aspects of agents or their reflective and goal-oriented attitudes (Bauer & Muller, 2004; Iglesias *et al.*, 2000; Zambonelli *et al.*, 2001).

Consequently agent-oriented approaches were developed to provide support for modelling artifacts such as goals, intentions and organizations. Examples include the Gaia methodology (Wooldridge, Jennings. & Kinny, 2000), its extension ROADMAP (Juan, Pearce & Sterling, 2002) and the SODA methodology (Omicini, 2001). However, as the underlying conceptual models were proprietary, difficult to understand without a background in agent technology, and few, if any industrial-strength tools were available to reduce the complexity, purely agent-oriented approaches were not easily implemented and accepted by industry (Bauer & Muller, 2004).

As a result, a number of methodologies to extend and include agent-oriented features into OO approaches such as UML and RUP (Rational Unified Process) were developed (Bauer & Muller, 2004). Examples of these include Agent Modelling Technique for Systems of BDI Agents (Kinny, Georgeff &

Rao, 1996), TROPOS (Giunchiglia, Mylopoulos & Perini, 2002; Mylopoulos, Kolp & Castro, 2001), Prometheus (Padgham & Winikoff, 2003) and MaSE (Wood & DeLoach, 2000). Extensions to UML such as AUML (Bauer, Muller & Odell, 2001), and M-UML for mobile agents (Saleh & Elmorr, 2004) have also been proposed. OO methodologies do not address the different types of communication and the social relationships between agents or their autonomous and proactive behaviour (Bauer & Muller, 2004; Iglesias *et al.*, 2000; Zambonelli *et al.*, 2001). Accordingly, some trade-offs are required in the design process, particularly regarding the natural design of agent-based systems. Nevertheless, they offer the advantage of fitting easily into the OO paradigm and enable the development of design tools by extending existing OO tools (Bauer & Muller, 2004). The similarities between agents and the specialization from objects to agents are covered in section 3.4.2.

Other design approaches include patterns and components. A pattern is a reusable solution to a common problem, based on proven techniques and approaches from other developers, and taking relevant forces into account. Several types of patterns exist, such as analysis patterns, design patterns and process patterns (Ambler, 2001). Design patterns are defined as 'explicit formulations of good past software development experiences consisting of proven solutions to recurring problems that arise within some contexts and systems of force' (Tahara *et al.*, 1999). Various patterns have been presented for agent systems (Tahara *et al.*, 1999), multi-agent systems (*Agent Factory*, 2003; Cossentino *et al.*, 2002; Platon, Sabouret & Honiden, 2004) and mobile agent systems (Aridor & Lange, 1998; Kendall *et al.*, 2000). A possible structure for a pattern catalogue for agent-oriented patterns is also proposed (Lind, 2002b).

A component is a cohesive unit of functionality that can be developed independently, delivered and combined with other components to build a larger unit. A component is typically (though not necessarily) implemented as a collection of classes and has contractually specified interface(s), which define its access points. With a pure OO approach, software is built from a collection of classes, while with a component-based approach software is built from a collection of components (Ambler, 2001). In Kotz *et al.* (2002) Picco proposes structuring mobile agent systems as a set of mobility components in which mobile code and mobile agents interoperate with existing mechanisms, such as remote method invocation, wherever possible. The current features in mobile agent systems would also be available as a set of orthogonal components that could be plugged in by programmers to provide for example mobility, security or communication. This would provide much more flexibility than current mobile agent systems allow, but it could be a quite difficult challenge to convert mobile agent system features into standard, reusable, orthogonal components to be combined as needed. Composing agents and mobile agents from components, is explored in Errol, Lang and Levy (2000); Gray (2004); Jazayeri and Lugmayr (2000); Lauvset and Johansen (2002); Lee and Shepherdson (2003); Marques, Silva and Silva (2000).

3.2.4 The agent-oriented programming landscape

A programming environment is the collection of tools used in the development of software. In the context of agents, this is frequently called an agent toolkit. As pointed out earlier, mobile agents and agent-based systems need an environment providing services such as basic communication, the discovery and coordination of agents, security, and so forth. Agent developers use *development tools* and *programming languages* to implement agent systems. Development tools and programming languages, in turn, are based on *agent architectures* and different *agent theories*, which are rooted in disciplines such as sociology, philosophy, psychology and biology (Braubach & Pokahr, 2004).

3.2.4.1 Agent Theory

Agent theory concerns the use of mathematical formalisms to represent the properties or attributes of agents, to define how these attributes are related and to reason about them (Wooldridge & Jennings, 1995). Examples of agent theories cited by Wooldridge (2002) and Wooldridge and Jennings (1995) include intention logic and the BDI model of rational agency, while Braubach and Pokahr (2004) also includes agent-oriented programming and subsumption theory.

3.2.4.2 Agent Architectures

As indicated earlier, *agent architectures* provide software engineering models of agents (Wooldridge & Jennings, 1995) and are used to conceptualise the perception, reasoning and action components of agents in order to view agents as reactive/proactive entities (Flores-Mendez, 1999). An agent architecture therefore provides information about essential data structures, relationships and control flow between the data structures, the processes or functions that operate on these data structures, and the operation or execution cycle of the agent (Lind, 2002b; Luck *et al.*, 2004a; Wooldridge, 2002). In (Luck *et al.*, 2004a) four different types of agent architectures are pointed out, namely reactive (for example the subsumption architecture), deliberative (for example BDI) and hybrid agent architectures (for example TouringMachines) for single-agent systems, and distributed agent architectures (for example Contract Net Protocol) for multi-agent systems. Single-agent systems also use layered architectures (Mangina, 2002).

3.2.4.3 Development Tools

To implement agent systems, the design models developed during the design process should be translated into program code. This can be done in two ways. The design models can be expressed in enough detail to allow direct implementation in an existing language such as Java, or an agent programming language that provides the programming constructs to implement agent concepts can be used. In both cases, a programming environment should be provided to facilitate the conversion into program code and to implement the agent system (Dastani, 2004). It can be very difficult to build sophisticated software agents from scratch, as specialized skills and knowledge of agent architectures, communication technology, reasoning systems, knowledge representation, ACLs, and protocols are required. Both Hayes-Roth and Amor (2003) and Luck *et al.* (2004a) hence recommend that software developers without agent expertise use an agent construction toolkit to build software agents quickly and easily.

Agent toolkits deploy the infrastructure required to support agent applications and offer the necessary support for developing agents (Lind, 2001; Luck *et al.*, 2004a). This infrastructure typically includes the network-programming and agent communication, security and thread synchronization necessary to implement the agent system (Badjonski *et al.,* 2005). Agent toolkits thus provide the framework to implement agent system architectures, and in this way compose the agent programming environment. This may be seen as providing a kind of operating system for agents, layered over the standard operating system. Support may range from graphical development environments to management and monitoring services. Toolkits differ considerably in the manner they facilitate the development of an agent system. Some, such as Zeus, use a graphical development environment and require almost no need for code. Others, for example RETSINA, allow the use of a variety of languages (Java, C, C++, Python, Perl, Lisp), while still others use agent programming languages such as JACK and DECAF (Luck *et al.*, 2004a).

Accordingly, the actual form of toolkits also vary widely: from integrated development environments providing a graphical interface, to some form of middleware providing networking capabilities or sets of *API*s that a programmer can integrate into his/her own solution. Agent toolkits may typically offer (Luck *et al.*, 2004a)

- facilities to enable the development of individual agents and their interface to the environment – namely they allow the developer to construct a single agent and to specify how that agent can perform actions to influence its environment. This includes agent capabilities such as planning or reasoning as well as the agent architecture or agent model used to convey that;

- discovery, communication, ontologies and coordination mechanisms to provide coordination and communication between multiple agents with regard to both low-level and high-level services. Low-level services include generic services that any distributed system infrastructure requires, such as enabling middleware and basic security services. High-level services refer to requirements specific to the operation of an agent-based system, such as coordination mechanisms and complex security infrastructure;
- management services to monitor and debug agent applications; and
- software specifically aimed at assisting with the software development process, such as an integrated development environment, or a set of classes to be used in the construction of an agent.

3.2.4.4 Implementing Agent Systems

Agent systems are frequently implemented using OO programming, logic-based programming or a combination of logic and OO programming (Braubach & Pokahr, 2004). In Figure 3.4 Braubach and Pokahr (2004) provide a visual overview of the current agent programming landscape from the view of an agent developer. This is done by classifying some concrete agent architectures, languages and development tools.

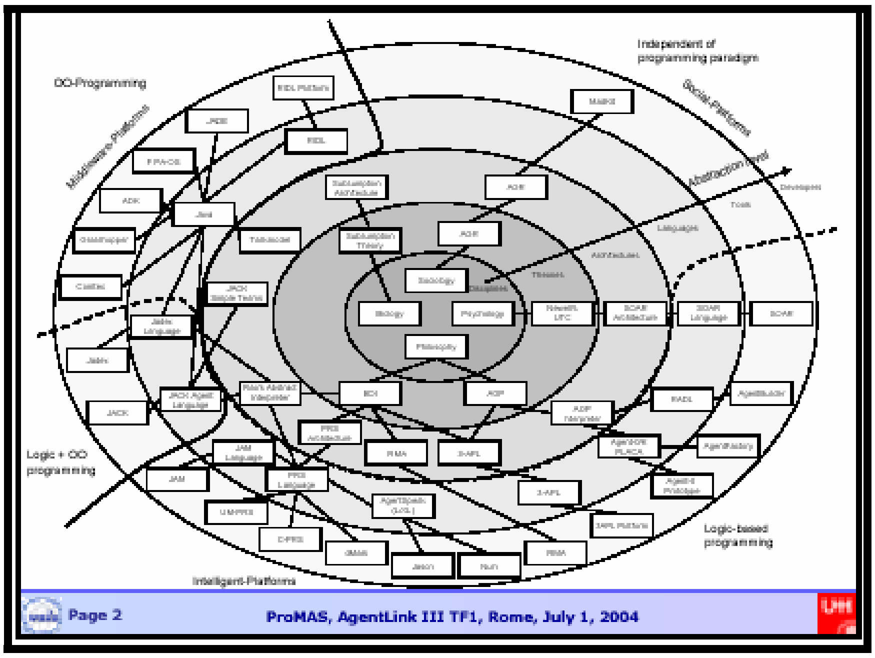


Figure 3.4 The agent programming landscape

This classification reveals two main platforms used for agent development:

- 'Middleware' agent platforms provide a conventional OO programming interface, and use a simple task-based agent model instead of a theoretically grounded architecture.
- 'Intelligent' or 'reasoning-oriented' agent platforms provide deliberative capabilities based on an agent model, such as BDI, and follow a logic-based approach.

Middleware platforms focus on FIPA-related[9] issues such as interoperability, security and maintainability, as well as infrastructure concepts such as white and yellow page services. Most middleware platforms intentionally leave open the issue of internal agent architecture and employ a simple task-oriented approach. The overall agent behaviour can then be taken apart into smaller pieces and attached to the agent as required. The tasks themselves can also be implemented in an OO language such as Java, which allows the software developer a gentler introduction to agents (Braubach *et al.*, 2004a). The majority of current mobile agent systems fall into this category and use a class library to provide facilities to support mobility (Luck *et al.*, 2004a).

Reasoning-oriented platforms, on the other hand, concentrate on the behaviour model of a single agent trying to achieve rationality and goal-directedness. Behaviour models are typically based on adapted theories from disciplines such as philosophy, psychology or biology. Behaviour models tend to become complicated and difficult to implement and use, especially when advanced AI and theoretical techniques such as deduction logics have to be employed to program the agents (Braubach *et al.*, 2004a).

Middleware agent platforms naturally allow agent-based programming, in contrast to reasoning-oriented platforms, which implement agent-oriented programming. On a reasoning-oriented platform, an agent system is often implemented with an agent programming language. *Agent programming languages* are often developed as an extension or specialization of OO languages and allow the programming of a system in terms of *some* of the concepts developed by agent theorists. They typically include some structure corresponding to an agent, as well as possibly some other attributes relating to agent concepts, such as beliefs and goals (Wooldridge & Jennings, 1995). AI techniques, such as a construct corresponding to a neural network, can also be included (Badjonski *et al.*, 2005).

[9]http://www.fipa.org

Examples are the Telescript language (White, 1994; 1997), the JACK Agent Language (an extension of Java) (*JACK*), AGENT0 (Shoham, 1993), PLACA (Thomas, 1995) and Concurrent METATEM (Fisher, 1994). Multi-agent programming languages should include programming constructs to implement the individual agents as well as programming constructs to implement their communications and interactions. Jadex, Jason and 3APL support the development of multi-agent systems (*Multi-agent Systemen*, 2004), while CLAIM (Fallah-Seghrouchni & Suna, 2003a; 2003b) is intended for multi-agent systems comprising intelligent mobile agents.

Many agent-based systems deployed in industry are implemented directly in *OO languages*. The close similarity between objects and agents (see section 3.4.2) makes it easier to implement agents in an OO language rather than in a language such as Prolog. The absence of a standard agent programming language and the environment provided by the Java virtual machine are also listed as some of the reasons for this (Dastani, 2004).

Certain agent toolkits, such as RETSINA and Zeus, allow agents to operate as a stand-alone program, while others, for example JACK, JADE, IMPACT and living markets, provide some kind of containing environment in which agents have to operate. Such a containing environment offers supporting services for all agents, provides a standardized approach for communications and appears to be the only way to support mobility for agents (Luck *et al.*, 2004a). The containing environment provides protection and allows access control to local resources, one of the features required from a mobile agent system, as indicated in 2.3.3.4.

A wide variety of agent construction toolkits, both in number and capabilities, is available. Given the lack of a standard agent construction toolkit, combined with the number of AOSE methodologies available, guidance in choosing the appropriate methodology and/or agent construction toolkit to develop an application should be sought. Though it falls beyond the scope of this dissertation, several researchers such as Lall (2005), Müller (1998), Sudeikat *et al.* (2004) attempt to provide guidelines for choosing the appropriate methodology and/or toolkit.

3.3 The OO paradigm

The OO paradigm is a development strategy based on the concept that systems should be built from a collection of reuseable components called objects (Ambler, 2001). In this paradigm a system is seen as a collection of interacting objects that models the interactions between objects to achieve the desired systems functionality (Ambler, 2001; Cloete & Gerber, 2004). Objects *do* things (have functionality) and *know* things (have data) (Ambler, 2001). The data or attributes that an object holds embodies the state of the object, namely what it knows. The operations on the data, or its functionality (its methods), indicates the object's behaviour, namely what it does/can do. This resemblance to actual entities provides objects with the ability to model elements of real systems as well as to serve as components in software systems (*Object-oriented programming and the Objective-C Language*, 1996).

In the object-orientation paradigm, a program consists of a collection or network of interconnected objects calling upon each other in order to solve a problem. Program design for an OO system consists of defining a hierarchy of classes that describe the relationship of the objects that will populate the solution program, as well as laying out the network of objects with their behaviours and patterns of interaction. There is no central control and the object network is activated by an external stimulus (often a flow of events). Each object plays a specific role in the program design and collaborates through messages with the other objects that it has a relationship with (*Object-oriented programming and the Objective-C Language*, 1996; Ambler, 2001; Horstmann & Budd, 2005; Voss, 1991).

Objects are often seen as 'actors' 'performing' their methods in the sense that each object plays a particular role within the overall program design. The objects therefore act as the tools or delegates executing the program's activity. Within its role an object acts fairly independently of the other parts of the program, since it is self-contained and to a certain extent can act on its own. However, despite the role possibly being multi-faceted and quite complex, the object still follows the 'script' programmed for it through its methods (*Object-oriented programming and the Objective-C Language*, 1996). Though UML and Java have introduced event-listener frameworks and other mechanisms to allow objects to be more active, traditional objects are still passive, as their methods can only be invoked under an external entity's thread of control (Odell, 2002).

OO programming itself involves three fundamental concepts: abstract data types, inheritance, and polymorphism. OO programming languages support the paradigm with classes, methods, objects and message passing. An *abstract data type*, called a class, generalizes a collection of similar objects and represents a template that can be used to create or instantiate similar objects. An object (instance of a class) encapsulates data and procedures in a single entity, and hides information by restricting access to it. Specializations of a class (the base class or superclass) can be defined through *inheritance* (derived classes or subclasses). A class can simultaneously be a derived class and a base class for its own derived classes, thereby specifying a hierarchy of objects. *Polymorphism* refers to the ability of similar objects to respond to the same message in different ways and enables objects to interact with one another without knowing their exact type (*Object-oriented programming and the Objective-C Language*, 1996; Ambler, 2001; Horstmann & Budd, 2005; Sebesta, 1999; Voss, 1991).

The relationships or interconnections providing the interaction *between* objects can take various forms:

- Inheritance - where a more specialized class (the derived class) inherits from a more general class (the base class). This is often described as an *is-a* relationship.
- Dependency - a class depends on another class if one of its member functions or methods uses an object of the other class in some way. This is often described as a *uses-a* relationship.
- Association - A class is associated with another class if the class has a data field (attribute) whose type is another class, namely one can navigate from objects of the class to objects of the other class. Associations are maintained with attributes to store the information necessary to maintain the relationship, and methods to keep the attributes current.
- Aggregation - Aggregation is a stronger form of association when a 'whole-part' relationship exists between classes. This is often described as a *has-a* relationship.
- Composition - Composition is a stronger form of aggregation in which a 'part' can only belong to one 'whole' at a given time, and the 'part' exists only while the 'whole' exists (Ambler, 2001; Horstmann & Budd, 2005).

In order to maintain relationships, objects and the information needed to maintain the relationships with which they are involved, are made persistent, by saving these to permanent storage in either an object database or a relational database.

3.4 Agent orientation as an extension of OO

Several researchers maintain that object-orientation fails to provide an adequate set of concepts and mechanisms for modelling complex systems (Booch, 1994; Huhns, 2004; Jennings, 1999a; Zambonelli *et al.*, 2001). Therefore, agent-oriented system development aims to simplify the construction of complex systems by introducing a natural abstraction layer on top of the OO paradigm composed of autonomous interacting actors (Braubach, Pokahr, Lamersdorf & Moldt, 2004b; Wooldridge & Jennings, 1999). In fact, agents provide a different and higher level of abstraction than objects, as the behaviour of agents more closely mirrors that of the people whose work they are assigned to carry out and/or support (Luck *et al.*, 2004a; Zambonelli *et al.*, 2001).

This section investigates how agent orientation extends OO by describing the historic development of programming paradigms and comparing objects and agents. Objects and agents are compared by identifying similarities between them, as well as indicating the manner in which agents enhance objects.

3.4.1 Historic development

The evolution of programming approaches is explained with the aid of Lind's (2001) three-step characterisation, as shown in Figure 3.5:

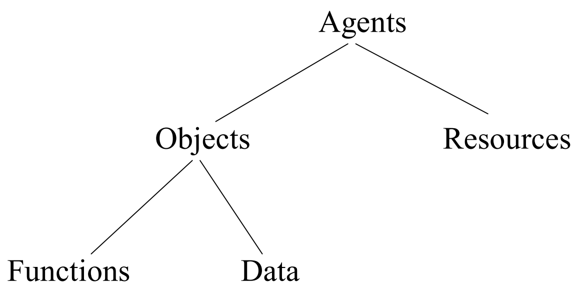


Figure 3.5 Lind's three-step characterization (Lind, 2001)

Initially the basic unit of software consisted of a complete program. A program was seen as a collection of *functions* executed in a pre-determined order to reach a specific well-defined goal. The programmer had full control and determined the behaviour of the program before it began execution. Accordingly, the program's state was the responsibility of the programmer and the system operator invoked it.

During the next development step (structured programming), a modular approach was followed to organize a program into smaller units of code that could be reused in a variety of situations. The subroutines and structured loops concerned were designed to have a high degree of local integrity. A program is hence interpreted in terms of its *data* and the *functions* used to manipulate the data. The state of the unit is determined by externally supplied arguments and it can only gain control when invoked externally.

Abstract data types created stronger and more explicit relations between data and functions, leading to OO programming. Here each *object* has local control over the *variables (data)* manipulated by its *methods (functions)*, enhancing the modular approach. In traditional OO, objects are still regarded as passive, since their methods are only invoked when some external entity sends them a message. Objects can be invoked at any stage during the program execution.

In the last step of the characterisation, *objects* are enhanced by the allocation of *resources* that can be used freely, such as computation time. This freedom in allocating resources provides autonomy to *agents*, and allows software systems to be perceived as a collection of active entities. For mobile agents, the ability to move themselves allows access to additional resources. Agents can use a variety of messages to interact with other entities, including method invocation, notifying, requesting and receiving messages from other agents and their environment. Each agent has its own thread of control, thereby localizing code, state and invocation. The individual rules and goals of an agent can determine when and how it acts, and little or no interaction is required to initiate an application. This view supports the principle of locality, since all the elements that combine to create the control-flow of a specific agent are grouped together. In an OO system, in contrast, the control-flow specification is distributed through the entire program code (Lind, 2001; Odell, 2002; Parunak, 2000). Multi-agent system focus on the cooperation between agents and explicit structures, mechanisms and protocols are used to implement this.

3.4.2 Agents and objects

The lack of a unanimously established definition of an agent gives rise to disagreement among practitioners and researchers on the difference between agents and objects (Luck *et al.*, 2004a; Odell, 2002; Wooldridge, 2002). This section attempts to clarify this by considering agents and objects from the viewpoints of the two paradigms (agent orientation and OO) and their implementation, in order to show how the agent orientation paradigm extends the OO paradigm. Agent orientation and OO are compared at two levels: during design and analysis, and during implementation. Table 3.1 provides a summary of this comparison, and points out both similarities and differences between agent orientation and OO.

3.4.2.1 Comparing OO and agent-oriented analysis and design

In both the OO and AOSE paradigms a system is regarded as a set of interacting entities working together to achieve a goal. Accordingly, during the analysis and design phases of both paradigms, these entities as well as their interactions and interrelationships are identified and defined. However, in agent orientation not only the individual responsibilities and corresponding tasks for agents are identified, but also the global responsibilities and the corresponding social tasks of the multi-agent system as a whole. This means that agents may pursue their own goals as well as a common goal. In general objects cooperate to achieve a common goal, but does not each have a goal of its own. Furthermore, in pursuit of the common goal(s) in a multi-agent system, emergence often ensues, which is not the case in an OO system. In a single agent system the focus will be on the individual agent acting autonomously, which also differs from the traditional view of an OO system in which the objects are not autonomous.

OO methodologies do not define explicit techniques for modelling the autonomous and proactive behaviour of agents, or for their social relationships and the various types of communication they employ. OO methodologies also offer no support for capturing the reactive and proactive behaviour of agents, or for maintaining a balance between the two (Zambonelli *et al.*, 2001). As a consequence, the resulting system architectures for OO systems and agent systems differ. Both agents and objects can use inheritance and aggregation to define their architecture (Shoham, 1993). A traditional OO software architecture expresses the functional dependencies or relationships between objects. An agent or multi-agent system architecture, though, models high-level dynamic interaction between agents, as well as agent interactions with the environment (Zambonelli & Omicini, 2004). Since no first-class notion of organizational structure exists in OO, OO approaches provide only minimal support for specifying and managing organizational relationships. Relationships are basically defined by static inheritance hierarchies (Jennings, 1999a; 2001). This is insufficient to denote organizational relationships between real-world entities. As discussed earlier, agents in a multi-agent system are regarded as a society and each agent in the multi-agent system can be viewed as both an individual software system and part of a society. Procedures for modeling these social relationships between agents have to be defined. In AOSE organizational relationships, such as collections of agents (subsystems), are explicitly represented in structures, and interaction protocols are used for grouping and disbanding sets of agents (Zambonelli *et al.*, 2001).

3.4.2.2 Comparing agents and objects at implementation level

As described in 3.3, an object is a high-level abstraction that encapsulates state (attributes or data) and behaviour (methods or operations on the data). Objects typically represent abstractions of things, or *passive* entities in the real world (Huhns, 2004; Lind, 2001; Travers, 1996; Tveit, 2001).

In contrast, an agent was earlier described as a software system that is situated in an environment in which it operates in a continuous Perceive-Reason-Act cycle. The ability of the agent to *choose* how to react or respond to perceived changes in the environment provides the agent with *autonomy*, allowing it to display *flexible* behaviour, thereby becoming *active*. Agents are therefore generally regarded as representing abstractions of intelligent beings, namely active entities, and need to support structures for representing mental components such as beliefs and commitments. These structures are not normally embedded in an object, even though it is possible to build systems that do integrate these properties (Flores-Mendez, 1999; Wooldridge, 2002).

At an implementation level encapsulation allows objects to hide or control their own internal state, but leaves them no choice as to whether or not to execute a `public` method when it is invoked by a message from another object. The object sending the message to invoke a method, in effect then controls the execution of the method. The fact that objects cannot choose whether or not to interact makes them passive. Therefore, though objects encapsulate state and behaviour realization (similar to agents), they do not encapsulate behaviour activation or invocation (Jennings, 1999a; 2001; Wooldridge, 2002). In contrast to traditional OO languages, agent programming languages typically include special features such as a structure corresponding to an agent, and possibly programming constructs to implement agent communication, interaction and cooperation. This allows the implementation of agents as active autonomous entities with the ability to choose whether or not, as well as how, to respond to a request from another agent (Flores-Mendez, 1999; Luck *et al.*, 2004a; Tveit, 2001; Wooldridge, 2002). Furthermore, since software agents each has its own thread of control, agents also have the ability to initiate action at any time and cannot be directly invoked like objects, thereby encapsulating *behaviour activation* in addition to state and behaviour realization (Luck *et al.*, 2004a; Odell, 2002).

When comparing agents and objects, autonomy and interaction are perceived as the two key elements that distinguish agents from traditional objects (Odell, 2002). To a certain extent the varying degrees of autonomy and interaction as described by Odell (2002) and set out in the following paragraphs (where only references other than Odell are indicated) also characterize the manner in which agent orientation *extends* OO.

Autonomy has two independent aspects: dynamic autonomy and nondeterministic autonomy. *Dynamic autonomy* in agents varies from them being simply passive to entirely proactive, since they can react to both specific method invocations and to observable events in the environment. Proactive agents poll the environment for events and messages to determine their actions. In contrast, despite mechanisms that allow objects to be more active (see 3.1), traditional objects are still regarded as passive, since their methods are invoked under a caller's thread of control.

Nondeterministic autonomy refers to the degree of *unpredictable* behaviour, which may range from absolutely predictable to entirely unpredictable. Though conventional objects do not have to be completely predictable, they tend to be more predictable since object classes are designed to be predictable to encourage reuse. Agents, however, are usually designed to base their behaviour on individual goals and states, combined with the input from ongoing interaction with other agents, which causes their behaviour to be relatively unpredictable.

Furthermore, at the time of design, it is not always known how and when an object's functionality will be used, or how this will affect its state, which can lead to unanticipated errors (Huhns, 2004). Agents, however, are *designed* to handle unexpected events and, owing to their autonomy, have the flexibility to react or respond appropriately to perceived changes in the environment (Jennings, 2001; Luck *et al.*, 2004a). Agents therefore exhibit *flexible* (reactive, proactive and social) *behaviour*, while the standard object model does not provide for these concepts, even though it is possible to build systems to include these properties (Flores-Mendez, 1999; Wooldridge, 2002).

The more 'opaque' view of encapsulation in the agent-based approach also renders agent behaviour unpredictable, for two reasons:

- The requested behaviours that an agent may perform may not be known within the active system, since an agent can employ mechanisms such as publish/subscribe, protocol registration, 'yellow page' or 'white page' directories or broker agents to access methods or services. In OO, in order to code, the programmer needs to know what interfaces are supported by an object.
- Agent communication is usually asynchronous, with no predetermined flow of control from one agent to another, since an agent may initiate action at any time, and not only in response to a message. Agent languages therefore need to support parallel processing to enable asynchronous messaging and event notification. OO languages do not usually support asynchronous messaging and event notification.

Interaction, namely the ability to communicate with the environment and other entities, can also be expressed in degrees. It varies from the most basic level, which is object messages or method invocations, to agents reacting to observable events in the environment, and at the highest level, multiple, parallel interactions with other agents in multi-agent systems.

Both objects and agents use message-passing to communicate with other objects or agents. For objects, message-passing is usually simply method invocation, though they can also communicate by changing data members to which they have access rights, or that fall within their scope (Venter & Barrow, 2003). In contrast, while agent-based communication can also use method invocation, agents distinguish between different types of messages. Often these messages are modelled as speech-acts in an ACL, using complex protocols to negotiate. As mentioned earlier, objects thus communicate at a purely syntactic level, while agents interact at a knowledge level (Iglesias *et al.*, 2000; Jennings, 1999a; Luck *et al.*, 2004a; Shoham, 1993). In an agent-based environment, agent public services and policies can be advertised through mechanisms such as publish/subscribe, protocol registration, page directories or broker agents. Agents also analyze the messages and can decide whether or not to execute the requested action.

Agents can have long-term conversations (negotiations) and associations, as well as participate in multiple conversations simultaneously by means of multiple threads. In such a situation each conversation would needs its own identity, while a unique message destination or identifier can also be used to distinguish threads of conversations. While conventional OO languages and environments cannot easily support such situations, objects could be used for elements of agent conversation where little or no autonomy or interactive capability is required.

Agents may also use third party interactions to communicate, by employing for example a broker that accepts a request and delegates it to a service provider, or an anonymizer that hides the identity of a requester from the service provider. Strongly typed object systems find it difficult to support these situations (Odell, 2002).

In a distributed system, client and server objects generally reside on different computers while OO middleware and component transaction monitors provide the interacting facilities required to present the system as an integrated whole. Similarly, software agents can reside on different platforms and require interacting facilities in order to enable them to communicate and cooperate (Brantschen & Haas, 2002).

Because of the interactive and autonomous nature of agents, applications can be launched with little or no physical interaction. No centralized thread or top-down organisation is required, since agent systems can organize themselves (Luck *et al.*, 2004a; Odell, 2002). Furthermore, since agents have at least one thread of control each, agent systems and especially multi-agent systems are inherently *multi-threaded*. Even though some programming languages provide constructs for concurrency in OO programming, in the standard object model a system has a single thread of control dispersed throughout the entire program code (Flores-Mendez, 1999; Luck *et al.*, 2004a; Wooldridge, 2002).

In summary, though objects and agents are both high-level representations of software components by means of their states and behaviour, *conceptually* agents represent a higher level of abstraction. Agents differ from traditional objects in the following ways:

- They represent *flexible, active, autonomous* entities since they may decide not to execute a request, whereas traditional objects are passive and have no choice as to whether or not to execute an invoked method.
- Agents are *interactive* and able to use a variety of forms of communication such as method invocation, ACLs, advertising services, etc. To communicate, traditional objects, in contrast, mainly use messages and invoke methods.
- An agent system is usually multi-threaded and has *no centralized control*, while the standard OO system has a single thread of control.

In conclusion, while agent orientation and OO share several similarities, agents are regarded as extending objects to a higher level of abstraction owing to their autonomous, flexible nature. Though agents can be implemented as objects and can benefit from using OO technologies, the developer should keep agent orientation in mind while implementing agents as objects.

Element of comparison	Similarities	Differences	
		OO	Agent orientation
Entity		Object: high-level abstraction of a software component encapsulating its data (state) and methods (behaviour). Represent abstractions of things, namely passive entities	Agent: software system situated in an environment in which it operates in a continuous Perceive-Reason-Act cycle Represent abstractions of intelligent beings, namely active entities.
System view during analysis and design	System regarded as a set of interacting entities working together to achieve a goal. Entities, their interactions and interrelationships are identified and defined.	Objects cooperate to achieve a common goal, but each does not have a goal of its own.	Agents have individual goals as well as a common global goal.
Architecture	Both can use inheritance and aggregation to define their architecture.	OO software architecture expresses the functional dependencies between objects or components. Relationships are basically defined by static inheritance hierarchies, though reflection makes it possible to generate objects on the fly that inherit dynamically. Objects do not offer support for capturing the reactive and proactive behaviour of agents, or for maintaining a balance between the two	Agent or multi-agent system architecture models high-level dynamic interaction between agents, as well as agent interactions with the environment. Organizational relationships, such as collections of agents (subsystems) are explicitly represented in structures. Interaction protocols are used for grouping and disbanding sets of agents.
Implementation	Both objects and agents encapsulate state and behaviour realization	Objects *cannot choose* whether or not to execute a `public` method when it is invoked by a message from another object. Objects' methods are invoked when an external entity sends them a message Any object can invoke any publicly accessible method on any other object, so that behaviour activation cannot be encapsulated.	Agents are active autonomous entities with the ability to *choose* whether or not, as well as how, to respond to a request from another agent. Since software agents each has its own thread of control, agents have the ability to initiate action at any time. Agents cannot be invoked directly like objects, thereby *encapsulating behaviour activation* in addition to state and behaviour realization.

Element of comparison	Similarities	Differences	
		OO	Agent orientation
Autonomy: Dynamic autonomy		Traditional objects are regarded as passive since their methods are invoked under a caller's thread of control, though an object can also have its own thread and act autonomously within a system.	Dynamic autonomy can vary from them being simply passive to entirely proactive since they can react to both specific method invocations and observable events within the environment
Nondeterministic autonomy (degree of unpredictable behaviour)		Conventional objects tend to be more *predictable* since object classes are designed to be predictable to encourage reuse by means of inheritance.	Agents' behaviour is relatively *unpredictable* since they are usually designed to base their behaviour on individual goals and states, combined with the input from ongoing interaction with one another.
		Unexpected behaviour causes error messages.	Flexibility to react or respond appropriately to unexpected behaviour.
Interaction, namely the ability to communicate with the environment and other entities	Both objects and agents use message-passing to communicate.	Message-passing = method invocation or changing data members to which it has access rights / that falls within its scope	Different types of message-passing: method invocation, ACL, yellow page/white page directories, protocol registration, publish/subscribe to noticeboards, third-party interactions, for example broker agent.
		At syntactic level	At knowledge level
		Synchronous	Asynchronous
			Long-term conversations (negotiations) Multiple conversations simultaneously
Distributed systems	Agents, as well as client and server objects, generally reside on different computers and require interacting facilities.	Though OO systems can be multi-threaded, in the standard object model a system has a single thread of control dispersed throughout the entire program code.	Since agents each has at least one thread of control, agent systems are inherently *multi-threaded*. No centralized thread or top-down organization is required since agent systems can organize themselves. Applications can be launched with little or no physical interaction.

Table 3.1 Comparison between objects and agents

3.5 When to use agent orientation

This section investigates the conditions recommended for agent orientation, and indicates how mobile agents fit into this context.

Various researchers (Parunak, 2000; Weyns *et al.*, 2004; Wooldridge, 2002; Wooldridge & Jennings, 1999) indicate that autonomous agents and multi-agent systems seem to be applicable to systems in which:

- the environment is *open*, or at least highly *dynamic*, *uncertain* or *complex*;
- data, control, expertise or resources are *distributed*;
- agents offer a natural metaphor for delivering system functionality, namely for applications that can be *naturally modeled* as societies of interacting autonomous entities;
- a number of *legacy* systems have to be integrated or work together.

Closed agent systems are defined as systems designed by one design team for one corporate environment where agents share common goals in a single domain. *Open agent systems*, in contrast, may include more than one design team and multiple heterogeneous environments as well as a number of agents, which do not necessarily share common goals. A mobile agent system, in particular, is deemed to be open if an unknown agent can enter over the network (Picco, 2001). Because of their very nature, open systems are often *complex*.

Software engineers apply three tactics to assist in managing complexity:

- Decomposition to divide a large problem into smaller, more manageable parts with limited scope so that these can be handled in relative isolation.
- Abstraction to emphasize some details or properties, while others are concealed, once again to limit the scope of the problem.
- Organization to define and manage the interrelationships between various problem-solving components, as this can group smaller-grained components together so that they can be treated as a higher-level unit.

Complex systems thus consist of a number of related subsystems organized in a hierarchical way. Each component in the system strives to achieve one or more objectives, while at the same time these subsystems work together to attain the functionality of the system. Each individual component should have its own thread of control (be active), as well as control its own choices and actions (be autonomous). To reach their individual and collective goals, the components have to interact. Since a complex system's inherent complexity makes it impossible to know about all interactions beforehand, components need the ability to initiate and respond to interactions in a flexible manner at run-time. Components in a subsystem thus correspond largely to the concept of autonomous agents, while subsystems correspond to agent organizations. Both autonomy and organization allow incorporating abstraction in the system. The interplay between subsystems and their constituent components also match high-level social interactions, in accordance with the agent-oriented approach. The changing sets of relationships between the components in a complex system, as well as treating a set of components as a single conceptual unit, also fit in with the agent-oriented mindset (Jennings, 1999a; 1999b; 2001). The telecommunications industry provides an example where mobile agents are applied to manage the system load in complex distributed networks with differentiated components. To adapt the system load, mobile agents are used to move the functionality around the physical network according to changing traffic patterns (Gray, 2004; Luck *et al.*, 2004b; Manvi & Venkataram, 2004).

As highly *dynamic* systems change frequently, the decentralized and modularized nature of agents minimizes the impact that modifications to a module (agent) may have on the rest of the system. In this situation mobile agents can be used to update components and for quality of service negotiations. E-commerce and telecommunications applications are examples where this is applicable (Gray, 2004; Luck *et al.*, 2004b; Manvi & Venkataram, 2004). Another example of dynamic systems occurs in mobile computing, where dynamic adaptation to the various forms of user connectivity is frequently required. Mobile agents support that as well as the asynchronous execution of client requests, weak connectivity and disconnected operations common in mobile computing (Samaras, 2004). Similarly, a pervasive system needs to be context-aware in order to adapt dynamically to changing conditions. Mobile agent technology can assist in this regard by allowing intelligent agents to move to different locations and retrieve context-related information from there (Zaslavsky, 2004).

An *uncertain* environment is ill-structured, because it is under-specified. In such an environment agent orientation allows the analyst to design the classes or agents that *generate* a given domain structure rather than the structure itself. This will extend the useful life of the system as well as reduce the cost of maintenance and reconfiguration (Parunak, 2000). Here mobile agents offer considerable advantages for network management and information retrieval. Mobile agents could search for information and services in an unstructured network, as well as provide distributed access to Internet databases, including distributed retrieval and filtering of information, thereby reducing network load. They can also be used to disseminate information and monitor distributed processes (Samaras, 2004).

The autonomous, flexible, pro-active nature of agents suits applications that can be decomposed into stand-alone processes, where each process executes without continuous direction from another procedure (Jennings, 2001; Parunak, 2000). This is frequently the case in systems where *data, control, expertise* or *resources* are *distributed*. An example of such an environment is Grid computing, an emerging computing model where heterogeneous resources (based on different platforms, hardware/software arcitectures and computer languages), located in different places and belonging to different administrative domains is shared over a network using open standards (*Grid computing,* 2006; Luck *et al*., 2003, 2004b). It has already been shown that mobile agents are well-suited to such an environment.

Agents are modular entities, and each agent has its own set of state variables, which, though distinct from those in its environment, are linked via the agent's input and output mechanisms to a subset of the environment's state variables. An industrial entity with a well-defined set of state variables that are distinct from those in its environment, and clearly identifiable interfaces, is therefore suitable for implementation as an agent (Parunak, 2000). This enables agents to represent distinct individuals and organizations, as well as groups, which interact at a knowledge level (for example to negotiate) to reach their individual and common goals (Jennings, 2001). Such an application can be *naturally modelled* by a society of stationary and mobile agents.

The recommendations by Whitestein Technologies (*Whitestein Technologies*) for using the recently released AML (Agent Modelling Language), spell out the conditions appropriate for implementing an application as an agent system even more clearly. AML is specifically designed for building models of systems that implement multi-agent system concepts, and can be used to build models that

- consist of a number of autonomous, concurrent and/or asynchronous (possibly proactive) entities;
- consist of entities that are able to observe and/or interact with their environment;
- use complex interactions and aggregated services;
- use social structures; and
- represent mental characteristics of systems and/or their parts (*Whitestein Technologies: Methodology*, 2005; Cervenka, Trencansky, Calisti & Greenwood, 2005).

These characteristics all correspond to those present in a society of agents.
Legacy (non-agent) software can be integrated into an agent system by placing wrapping software around the legacy code to present an agent interface on the outside to other software components. On the inside, the wrapper performs a dual translation process to map external requests from other agents into calls in the legacy code, while converting the legacy code's external requests into an appropriate set of agent communication commands (Jennings, 2001). While wrapping software around legacy (non-agent) software could be used to convert the legacy software into a stationary agent, it would typically not be possible to convert the legacy software into mobile agents.

Agents and multi-agent systems, however, do not provide a 'magic wand'. Wooldridge and Jennings (1999) point out a number of pragmatic pitfalls in six areas, namely political (the corporate environment), management, conceptual, analysis and design, agent level and society level. Political pitfalls include overselling agents and becoming dogmatic about agents by insisting on an agent system, regardless of whether it is suitable or not. Management pitfalls involve insisting on an agent system without having clear goals, or requiring a generic solution. Agent architectures and multi-agent systems are in fact suited only to certain types of conditions, such as those discussed in the previous paragraphs. Conceptual dangers consist of regarding agents as a silver bullet to provide order-of-magnitude improvements in software development, forgetting that agents are software that needs to be implemented, as well as overlooking multithreading as an inherent part of agent systems. Omitting to incorporate related technology, concurrency and legacy software during analysis and design presents pitfalls. Agent-level traps include developing one's own agent architecture from scratch, using too much or no AI. Society-level pitfalls include viewing everything as agents, having too few agents, implementing one's own infrastructure instead of using an existing platform, allowing agents to interact too freely and suffering lack of structure in the agent society.

Jennings (1999a) discusses problems inherent to agent orientation owing to the nature of agent-oriented software. Designing software able to maintain a balance between the reactive and proactive behaviour of agents is difficult, since agents have to pursue their own goals while constantly interacting with their environment. Such a balance requires context-sensitive decision-making. This implies that which goals will be followed under which circumstances, as well as which methods will be used to achieve this, is to a degree, *unpredictable*. Furthermore, since agents are autonomous, the patterns and consequences of their interaction are also uncertain, because the number, pattern and timing of interactions cannot be predicted in advance. Emergence, too, results in behaviour that cannot be deconstructed only in terms of the behaviour of individual agents.

As discussed above, agent orientation can be used to develop applications that are inherently distributed, open, large and heterogeneous. The benefits mobile agents hold for these types of applications have been indicated. Examples of such systems can be viewed in Manvi and Venkataram (2004), Satoh (2004, 2005), Tozicka (2003) and Zaslavsky (2004).

3.6 Conclusion

Just as many systems may be understood as a collection of interacting but passive objects, many other systems may naturally be understood and modeled as a collection of autonomous interacting agents. The first adheres to the OO paradigm and the second to the agent orientation paradigm.

The extension from OO to agent orientation shows that agents and objects are indeed different concepts. Nevertheless, AOSE can benefit from OO technologies and approaches, while supplementing the shortcomings of OO modelling as seen in 3.2.3.

Since agent orientation is a relatively young paradigm, and AOSE is still in its infancy, a considerable number of methodologies are in the process of being developed for agent systems. In this chapter it was seen that agents can be implemented using a variety of paradigms and technologies such as object technology, logic programming and component-based programming as well as specific agent technologies, for example agent-oriented programming. Agent development can be done via middleware platforms, typically by using OO languages, or by means of reasoning-oriented platforms following a logic-based approach. Systems developed with OO languages appear to be more agent-based, while systems developed using agent-oriented languages are naturally more agent-oriented, since agent-oriented languages already include structures and statements intended to be used for agent orientation.

Many of the traits deemed to indicate the use of agents and multi-agent systems, such as being open, dynamic, uncertain, complex, distributed and containing legacy systems that have to be incorporated, also apply to the Internet. Consequently, despite the problems described in section 3.5, factors such as the current computing trends indicate that mobile agents and multi-agent systems do have a place in the contemporary computing milieu. Mobile agents in particular hold significant benefits for Internet-based applications as well as for ubiquitous (pervasive) and mobile computing applications. Furthermore, current computing trends tend towards integrating mobile agents into multi-agent systems and/or endowing them with intelligence, in order to assist with pervasive computing and applications such as mobile computing, telecommunications and Web Services. This indicates that mobile agent programmers should take cognisance of agent orientation. As has been seen, agents and objects are different concepts. Therefore, to achieve the full potential of mobile agents in the current programming environment described above, they should be implemented according to the proper principles involved in agent programming.

In Chapter 4 a framework for constructing mobile agent system by following an agent-oriented approach is presented.

CHAPTER 4

4 A FRAMEWORK FOR CONSTRUCTING A MOBILE AGENT SYSTEM

4.1 Introduction

As described in previous chapters, building a commercially acceptable mobile agent system is a challenge, especially so for novice mobile agent programmers. This is even more true if such systems are to be constructed quickly and efficiently to gain commercial acceptability. One of the main reasons for this complexity is that so few guidelines are available on this topic. To concentrate on the research problem of this dissertation, it is therefore also necessary to concern oneself with what the architectural requirements for designing and developing mobile agent systems are. Addressing this concern will inevitably enable programmers to identify and include different mobile agent characteristics, with maximum reuse of available technologies and architectures. The purpose of this chapter is therefore to describe the architectural components within a generic mobile agent system model. These components represent the essential aspects of a mobile agent system in a simplified, general but clear way. In contrast to the detail on different characteristics and specific design issues of mobile agents, presented in Chapter 2, this chapter only establishes an architectural basis for a mobile agent system. A clear understanding of the architectural components that are involved is a constructive tool to a programmer or researcher who enters the field for the first time. As a tool it directs the broad and essential structures in which the anticipated system should be designed. The architectural model is also significant because it provides a benchmark for researchers and developers to measure the capabilities of mobile agents created by using commercially available tool kits.

In the quest to establish such an architectural model, various mobile agent implementations were studied, and features and characteristics that are embedded in these agent systems were extracted. In a further comparative study that was conducted, design and development guidelines were extracted. These guidelines are integrated into the architectural model that represents the essential features of a mobile agent system in the spirit of agent orientation.

Section 4.2 describes the architectural design for a number of existing mobile agent systems used to identify typical features and characteristics in a mobile agent system. Section 4.3 presents the proposed architectural model and section 4.4 compares it to one of the existing mobile agent systems described in section 4.2, in order to point out potential problems novices may experience as a result of missing layers in an architectural design. Section 4.5 discusses the significance of the architectural model and section 4.6 provides the conclusion.

4.2 Mobile agent implementations

In order to establish an architectural basis for the proposed architectural model, relevant architectural components are extracted from the available standards for mobile agent systems (see 2.3.4), and combined with features and characteristics that are embedded in a number of existing mobile agent systems. These are then integrated with design and development guidelines, in conjunction with the required paradigm shift to agent orientation, to provide a comprehensive model representing the essential features of a mobile agent system. This section discusses some of the existing mobile agent systems used to identify features and characteristics of mobile agent systems. Other systems studied include Dale (1997), Feridun and Krause (2001), Gschwind *et al.* (1999), Johansen *et al.* (2002), Kendall *et al.* (2000), Tripathi *et al.* (2001).

Four mobile agent platforms were selected to include in this discussion. Four of them, SMART (Scalable Mobile and Reliable Technology), D'Agents, Grasshopper (currently integrated with PLATIN as enago from IKV++ [*Master Consortium*]) and Aglets, are publicly available. Aglets is arguably the best known of all mobile agents, is implemented in Java and is MASIF-compliant. Grasshopper is the first mobile agent platform that was based on OMG MASIF and FIPA standards, and is also implemented in Java. SMART is implemented in Java, is MAF[10]-compliant and incorporates additional features such as dynamic aggregation and network class loading. D'Agents (formerly Agent Tcl) supports multiple languages (Tcl, Java and Scheme).

Only the architecture of each of the four platforms is described briefly. The common features from these models are extracted, and placed into the perspective of the mobile agent standards as described in Chapter 2. This comparison then leads to the architectural model proposed in 4.3.

[10] The common Mobile Agent Facility (MAF) is the predecessor of MASIF.

4.2.1 SMART

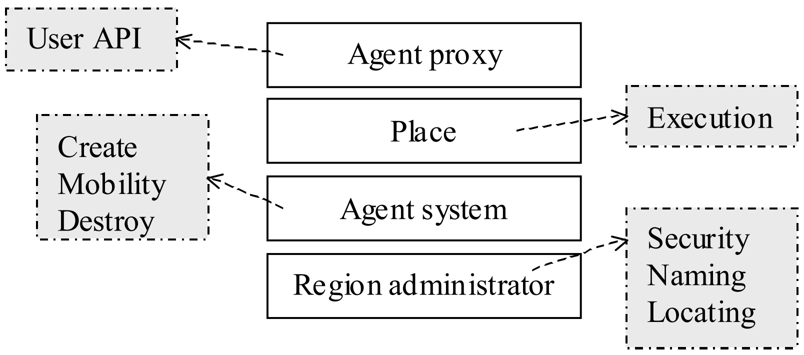


Figure 4.1 SMART architecture

Wong *et al.* (2001) developed a MAF compliant mobile agent platform, called SMART. This architecture consists of four layers built on a Java virtual machine (JVM). Figure 4.1 is a simplified depiction of SMART.

At the lowest layer the *region administrator* manages and enforces security policies on a set of agent systems. The *region administrator* uses a *finder module* to provide naming services to the region administrator and also to the layers above. In the second-lowest layer, the *agent system* layer, mobile agents can create, migrate and destroy themselves. The third layer from the bottom forms the execution environment and contains one or more *places*, which may exist for different execution contexts. In the topmost layer, the *agent proxy* provides the mobile agent API for applications written in SMART.

4.2.2 D'Agents

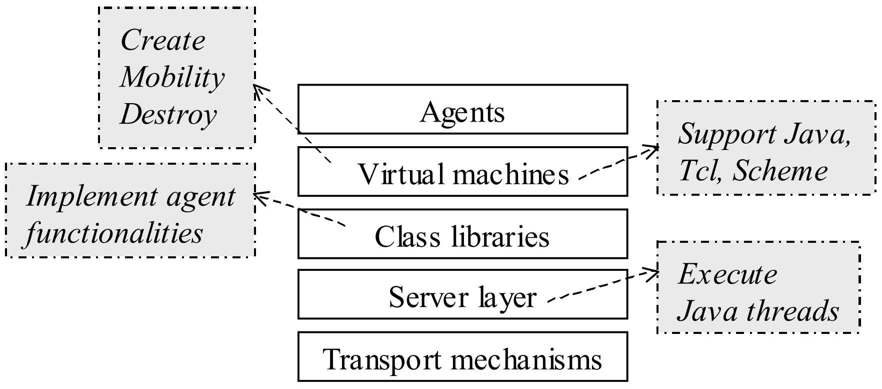


Figure 4.2 D'Agents architecture

Gray *et al.* (2002) describe the D'Agents (formerly known as Agent Tcl) mobile agent system, which was developed to support distributed information retrieval and to characterize the mobile agent performance space. In Figure 4.2 a simplified depiction of the D'Agents' architecture is shown.

The D'Agents' architecture has five levels. At the bottom level, TCP/IP is used to provide transport mechanisms. The second layer defines a *server layer* to be implemented on all hosts to accept incoming mobile agents. The server is a multi-threaded process and executes multiple agents as threads inside a single process, while each agent is executed inside its own (Unix) process. The next layer holds shared C++ libraries that implement agent functionality. The fourth layer provides the execution environment for each of the three supported languages (Java, Tcl and Scheme). Each such execution environment includes the interpreter/virtual machine for the supported language, stub routines allowing the agent to invoke functions in the C++ library, a state-capture module to support mobility, and a security model to enforce resource limitations. The agents themselves are defined in the top layer.

4.2.3 Grasshopper

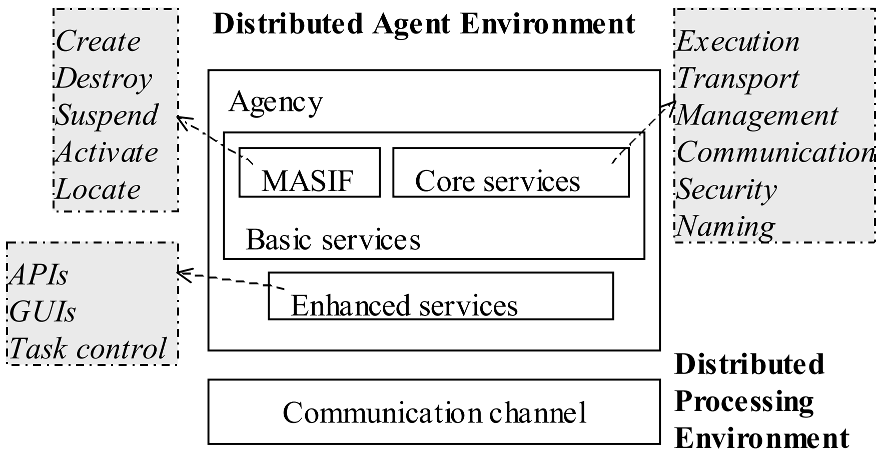


Figure 4.3 Grasshopper architecture

Grasshopper is a MASIF-compliant mobile agent platform supporting the development and execution of mobile agents. The Grasshopper architecture (Pavlou, 2000) distinguishes between two layers. The *distributed agent environment* resides on the top layer, while the bottom layer consists of the *distributed processing environment*. A host in the distributed agent environment typically includes an *agency* that has access to basic services as well as advanced/extended services. The basic services include a MASIF component with the *MAFFinder* and *MAFAgentSystem* interfaces as well as a number of core services. The core services include execution, transport, management, communication, security and naming mechanisms. Extended functionality includes adapter interfaces for external hardware/software, task control functions (for example itinerary), and application-specific graphical user interfaces (GUIs).

The *distributed processing environment* on the bottom layer is structured into regions (optional), agencies and places. Regions provide naming services and facilitate the management of distributed components. Grasshopper uses both stationary and mobile agents. The mobile agents migrate autonomously between different locations, while the stationary agents mostly reside permanently in their creation agency, offering services to other agents or entities. An agency provides the runtime environment for mobile agents, which offers the basic platform functionality that includes agent life cycle management, agent transport, communication, persistence and security. Each agency runs on its own JVM and consists of one or more places. In the Grasshopper architecture, a *place* that is residing in an agency is considered to be a grouping of agents that provide the same functionality. A region facilitates the management of agencies, places and agents as well as the tracking and finding of agents. Each region therefore contains a region registry that acts as a directory service, which also provides various lookup operations and search criteria. As can be seen, places, agencies and regions are used to organize agents in groups. Figure 4.3 is a simplified depiction of Grasshopper.

4.2.4 Aglets

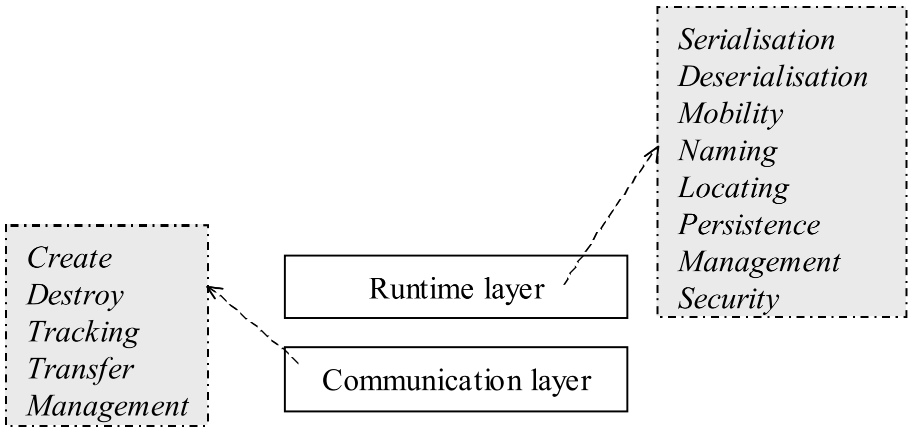


Figure 4.4 Aglets architecture

The Aglets initiative (Lange, 1997) is probably one of the best-known mobile agent projects, where mobile agents are referred to as 'aglets'. The Aglets architecture consists of two layers and two APIs that define interfaces for accessing layer functions.

The *runtime layer* (top layer) consists of a core framework and subcomponents to provide the following mechanisms fundamental to aglet execution: serialization and deserialization, class loading and transfer, reference management and garbage collection, persistence management, maintenance of byte code, and protecting hosts and agents (aglets) from malicious entities. These services are defined through the API on this layer.

The *communication layer* (bottom layer) uses a communication API that defines methods for creating and transferring agents, tracking agents and managing agents in an agent-system-and-protocol-independent way. The communication API is derived from the MASIF standard. The Aglets architecture uses the Agent Transfer Protocol (ATP), modelled on the HTTP protocol, as the default implementation of the communication layer.

The Aglets system uses the following life cycle: A new aglet must first be instantiated. This is done by either creating a new instance or cloning an existing aglet. Once created, an aglet object can be dispatched to and/or retrieved from a remote server, deactivated and placed in secondary storage, then activated later. An aglet can dispatch itself to a remote server, which causes the aglet to suspend its execution, serialise its internal state and byte code into the standard form and then be transported to the destination. On the receiver side, the aglet is reconstructed according to the data received from the origin, and a new thread is assigned and executed. Figure 4.4 is a simplified depiction of the Aglets architecture.

4.2.5 Essential features in mobile agent systems

As shown in Chapter 2, a mobile agent system provides the environment mobile agents need to exist and function. This environment consists of agent clients and agent servers. An agent client provides the API that allows a user to create and control agents. Agent servers provide the runtime environment for agents to execute in as well as several other functions in support of the mobile agent. Chapter 2 also indicated that mobile agent systems should exhibit the following features to achieve functionality: agent mobility, naming services, communication, controlled access to local resources, fault-tolerance/persistence, interoperability, agent management and control, and security. These requirements, the architectural components identified in each of the four platforms described in sections 4.2.1 to 4.2.4, and the mobile agent standards provided by FIPA and MASIF are summarized in Table 4.1. Where an architectural component corresponds to a feature identified in Chapter 2, only the name of the component as described in the relevant architectural model itself, is shown. However, some platforms offer features that can apply to more than one of the features indicated in Chapter 2. These are grouped towards the end of the table and all the features indicated in Chapter 2 that apply are given.

As can be seen in Table 4.1, none of the four mobile agent platforms completely satisfies the features required by a mobile agent system as identified in Chapter 2. Grasshopper's architecture satisfies most requirements, but, as can be seen from its depiction in both Figure 4.3 and Table 4.1, components are frequently located *within* several layers of other components. Furthermore, the Grashopper architecture consists of two layers that are mapped onto each other. This makes it difficult for a novice to form a broad overview, as well as to recognize individual components. The FIPA and MASIF standards also do not satisfy all the requirements listed.

It is clear that the classification of features provided in Chapter 2 is not fine-grained enough to describe all the essential components in a mobile agent system. The proposed generic mobile agent system architectural model described below includes all the components listed in Table 4.1, with the exception of an agent model and explicit reference to interoperability. As mentioned before, interoperability refers to interoperability between different mobile agent platforms and integration with existing applications. As can be seen from the MASIF standard, this covers a number of areas, namely a standard way of managing agents, a common mobility infrastructure to allow agents to communicate and visit other systems, a standardised syntax and semantics for naming services and a standardised location syntax for finding agents. Accordingly, the relevant components in the proposed architectural model handle these aspects. The proposed architectural model thus includes the requirements indicated in Chapter 2 and all the components available in the platforms discussed, as well as those identified by FIPA and MASIF. This provides an extensive overview of the environment in which a mobile agent exists and operates.

However, as indicated in Chapter 3, in agent orientation a multi-agent system focuses on addressing the *organization* and *coordination* of agents. The proposed architectural model therefore *enhances* the environment derived from Table 4.1 by adding a *social component*. The task of the social component is to control the organization and coordination of agents, as well as to enforce the resulting global social laws and conventions in a multi-agent system.

Though Grasshopper does offer a facility to organize mobile agents in groups by means of places, agencies and regions, no provision is made for explicitly coordinating the mobile agents, as is required in a multi-agent system. SMART uses a *region administrator* to enforce security policies on a set of agents, which hints at applying global laws, but does not provide a mechanism to organize agents into groups. The proposed architectural model thus portrays a complete view of the architectural components required to implement a mobile agent system in agent orientation. The next section presents the proposed architectural model.

Features required by mobile agent systems (from Chapter 2)	**SMART**	**D'Agents**	**Grasshopper**	**Aglets**	**MASIF**	**FIPA**
Agent client to provide agent management and control	Agent proxy		Enhanced services inside agency in distributed agent environment	Communication layer	Authority	
An environment that acts as interface between agent and server	Agent system	Server layer	Agency	Communication layer	Agent system	Management actions to support mobile agents
Constrained access to local resources	Places	Virtual machines	Core services inside basic services in agency in distributed agent environment Agency in distributed processing environment	Runtime layer	Place	
Agent mobility		Virtual machines Class libraries	MASIF inside basic services in agency in distributed agent environment	Runtime layer		Mobility protocols
Communication			Core services inside basic services in agency in distributed agent environment Agency in distributed processing environment			
Interoperability					MAFAgentSystem MAFFinder	
Fault-tolerance and persistence			Agency in distributed processing environment	Runtime layer		
Security	Region administrator	Virtual machines	Core services inside basic services in agency in distributed agent environment Agency in distributed processing environment	Runtime layer	Built-in Java constructs and CORBA services	

Features required by mobile agent systems (from Chapter 2)	**SMART**	**D'Agents**	**Grasshopper**	**Aglets**	**MASIF**	**FIPA**
Naming services	Region administrator		Core services inside basic services in agency in distributed agent environment MASIF component inside basic services in distributed agent environment Regions in distributed processing environment	Runtime layer Communication layer	MAFFinder	
Agent environment/constrained access			Use places, agencies and regions to organize agents into groups			
Mobility/Persistence				Serializing the agent before transmission and deserializing it on arrival at the remote host.	MAFAgentSystem	
Life cycle				Aglet life cycle		Mobility life cycle
Agent mobility		Transport mechanisms	Transport offered by agency in distributed processing environment	ATP		

Table 4.1 Features required by mobile agent systems

4.3 Proposed architectural model

As described in the previous section, the proposed generic mobile agent system architectural model provides a comprehensive view of the architectural components required to implement a mobile agent system, including the social characteristics required by multi-agent systems.

The proposed generic mobile agent system architectural model is depicted in Figure 4.5. A layered approach, organized according to the life cycle of a mobile agent, is followed. The layered approach isolates the individual components that each forms an essential part of the mobile agent system. In this way, while providing a broad overview of the important system elements during design, the approach also enables developers to focus on the tasks pertaining to a specific layer, which forms a functional unit. Furthermore, a layered model facilitates the development of layers in parallel, while implementing each layer independently also assists program maintenance, debugging and upgrading (Dale, 1997; Kendall *et al.*, 2000; Schoeman & Cloete, 2004).

Since the layers in the architectural model are organized according to the proposed life cycle of a mobile agent, this life cycle is described in section 4.3.1 before a detailed discussion on the individual layers in section 4.3.2 is conducted. Table 4.2 relates the execution states in the FIPA mobile agent life cycle to the life cycle and layers in the proposed architectural model. The detailed discussion of each layer includes guidelines to assist novice mobile agent programmers when developing a system. In Table 4.3, following after the detailed discussion, the purpose and significance of each of the different layers in the proposed architectural model is described.

4.3.1 Mobile agent life cycle

At the local host (see Figure 4.5), an authority (the person or organisation for whom an agent acts) uses the *authority API* to create one or more agents in the *agency*. Such an agent moves through the *execution layer* to the *social layer* where the global social tasks of the agent society are added to the mobile agent. In the *mobility* and *communication aspects* inside the *social layer*, standard mechanisms to ensure mobility and effective agent communication are added. Fault tolerance mechanisms are added at the *persistence aspect* in the *social layer*. At the *mobile agent management services layer*, the agent is serialised, before being encrypted at the *agent security layer*. At the *naming server layer*, the agent is named and registered for future reference. The location of hosts to be visited will also be

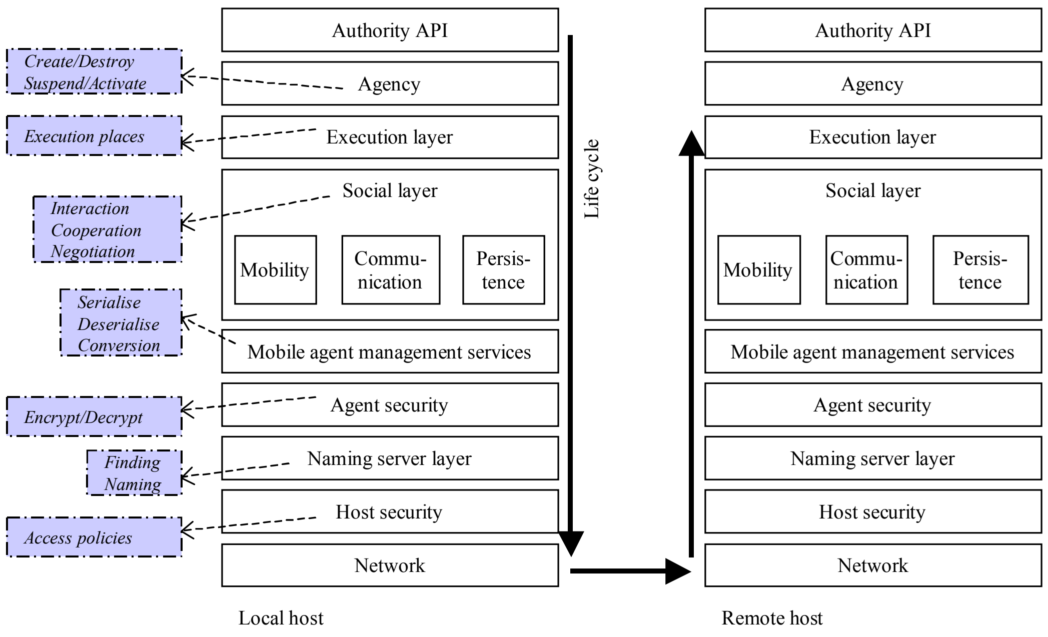


Figure 4.5 Proposed architectural model for mobile agent development

determined at the *naming server layer*. The agent receives credentials for access to other hosts at the *host security layer* before it is encoded in a suitable transport protocol at the *network layer* for transport through the network.

On arrival at a remote host, the *network layer* removes the transport protocol envelope and sends the agent to the *host security layer* where the agent seeks host access by submitting its credentials. If it is accepted, the host server registers the agent at the *server layer*, before the agent is decrypted at the *agent security layer*. The *mobile agent management services layer* is responsible for deserialising the agent and performing any conversions that may be necessary. The agent moves through the *social layer* to be executed at a *place* in the *execution layer*. The *social layer* enforces the social laws in the agent society during interaction between agents, the environment and other entities such as Web Services, in order to allow coordination, cooperation and negotiation. During execution, the agent will interact with the various aspects within the *social layer*: with the *persistence aspect* for storing information, with the *communication aspect* for communicating with other agents or entities, and with the *mobility aspect* for future migration requests. Interaction with the *persistence aspect*, the *communication aspect* and the *mobility aspect* are all subject to the social laws enforced by the *social layer*. Once executed, the agent follows the path downwards through the architecture in a similar fashion as at the local host. On arrival back at the *agency* in the local host, the owner of the mobile agent may use the *authority API* to dispose of the agent.

Table 4.2 relates the execution states in the FIPA mobile agent life cycle as discussed in section 2.2.4 to the life cycle and layers in the proposed architectural model. For each FIPA execution state the layer in the architectural model where it will occur, is shown.

FIPA execution state	**Layer in generic mobile agent system architectural model**
Initialized	The owner of the mobile agent creates the mobile agent by means of the *authority API* in the *agency layer*.
Active	The mobile agent is executed at a *place* in the *execution layer*.
Suspended	The mobile agent is suspended to prepare for migration by storing its state and intermediate results in such a way that it allows migration. During the preparation for migration, the *mobility aspect* in the *social layer* adds standard mechanisms to enable migration, while the *mobile agent management services layer* prepares (serializes) the mobile agent to be migrated.
Waiting	When the mobile agent is interrupted to wait for a specific event, it will temporarily pause its execution, but will remain in the *execution layer*.
Migration	The *mobility aspect* in the *social layer* adds standard mechanisms to enable migration, and the *mobile agent management services layer* serializes the mobile agent, but the actual migration occurs on the *network layer*.
Disposed	The owner of the mobile agent uses the *authority API* to dispose of the agent in the *agency layer*.

Table 4.2 FIPA mobile agent life cycle execution states in the generic mobile agent system architectural model

4.3.2 Model layers

4.3.2.1 Authority API layer

According to the MASIF definition the agent's authority is the person or organization on behalf of whom the agent acts. The *authority API* thus refers to the application interface that enables an authority to interact with and manage its mobile agents both locally and remotely. This includes tasks such as creating agents, communicating with them, and destroying them. Ownership and the relationship between owners and users are very important in mobile agent systems, and should be clearly indicated (Ashri & Luck, 2001). This takes place at the authority API layer.

Da Silva *et al.* (2000) remind developers that there are two interface types to design between an authority and its agency, namely an interface for from-authority-to-agents, as well as an interface covering from-agents-to-authority interaction. The first type of interface solves most of the interaction needs between the authority API and the agents. There are, however, also instances in which agents should take the communication initiative, such as reporting back functions, or when the agents encounter problems, in which case the second type of interface is required. Although there are many commercial agent development toolkits available, truly standardized system/independent agent APIs do not exist yet, and this necessitates further research and development (Luck *et al*., 2004b).

4.3.2.2 Agency layer

The term *agency* instead of *agent system* (as used by MASIF) is used to refer to the environment on a host where agents are created and terminated through an Authority API. The agent system is used to refer to the entire mobile agent platform stretching from the lowest to the highest level. To simplify agent design, the use of Wang, Hanssen and Nymoen's (2000) categorisation of agents into three basic types is suggested: namely *system agents*, *participation agents* and *user agents*.

The *system agents* take responsibility for administering the agency as well as other system functions residing on other layers of the architecture. These system agents can be further classified into

- *manager agents*, performing management tasks;
- *facilitator agents*, facilitating communication;
- *monitor agents*, logging events;
- *repository agents*, responsible for retrieving and adding information as well as querying repositories; and

- *interface agents,* providing the necessary API as well as interfacing with other entities and applications.

Participating agents provide system support for cooperative processes in execution places. These agents can be further classified into subclasses to simplify agent design, but the classification depends on the type of mobile agent application. This implies that a generic classification for this type of agent will not be sensible. For example, in an e-commerce environment one can create *negotiation* and *mediation agents*, but these classes of agents will not make much sense in an information retrieval application. Participating agents include *middle agents*: entities to which other agents advertise their capabilities (Flores-Mendez, 1999). Middle agents are neither requesters nor providers in a transaction, but assist in facilitating the process. Typical middle agents include *brokers* that receive requests and perform actions that combine services from other agents with their own resources; *matchmakers* and *yellow pages* that assist service requesters in finding service provider agents based on advertised capabilities; and *blackboards* that receive and hold requests for other agents to process.

User agents are those agents created by an authority for a specific purpose to act on behalf of its owner. From the description of the different types and classes of agents, it is clear that both system and participating agents are created as part of the agent system, since they support the operations of the entire system, whilst the user agents are created afterwards by the authority to perform a specific user task. Many programming patterns have been described to create and maintain these agents. For more information see Aridor and Lange (1998), Kendall *et al.* (2000), Singh, Sankar and Jamwal (2002), Silva and Delgado (1998) and Tahara *et al.* (1999).

4.3.2.3 Execution layer

The proposed model follows the MASIF definition of a *place* as the execution environment within a specific context where an agent executes. According to Da Silva *et al.* (2000), a place has two main objectives, namely to provide a conceptual and programming metaphor where agents are executed and meet other agents, and to provide a consistent way to define and control access levels and to control computational resources. Agent places can also have CORBA-support, facilitating external applications to communicate with the agents.

The execution environment is typically responsible for (Harrison *et al.*, 1995; Tripathi *et al.*, 2001)

- hosting the interpreter/virtual machine for the supported agent language;

- hosting stub routines allowing the agent to invoke different library functions;
- granting access to local resources in a selective manner;
- binding operating systems functions;
- binding to the network layer; and
- providing an environment where the agent can interact with or query its environment.

Although the agent may be executed in either machine or interpreted language, it is often preferable to express the agent in interpreted language since this supports heterogeneity better. It also has the advantage of late binding as well as the improved potential to add security mechanisms (Harrison *et al.*, 1995).

One of the major criticisms of mobile agents is that a guest agent can easily exploit its execution environment (Dale, 1997). *Constrained execution* is therefore quite important to control access granted to host resources. In this way the integrity of, and confidence in the host site is maintained. Take note that constrained execution in this context is not intended for security purposes. Access rights to the host (unlike access to host resources) are addressed on the host security layer.

The notion of places in the execution environment addresses the constrained execution concern to a certain extent, because it is easier to allocate access limitations on a place than to restrict the entire execution environment where the access limitations imposed on a place might be different for different places/agents. These access limitations on a place can be defined in quantifiable measures such as number of CPU cycles, number of files opened, etc. Both the limitations themselves, and the agent's awareness of such limitations will affect the manner in which an agent conducts itself, since the exhaustion of a constraint will typically result in the agent being terminated (Dale, 1997).

4.3.2.4 Social layer

Very few agent systems contain only one agent, and as mentioned previously, current trends tend towards incorporating mobile agents in multi-agent systems. In a multi-agent system *interaction* and *organization* are used to *coordinate* agent actions. This is necessary both to achieve the individual and common goals that agents in a multi-agent system pursue, and to prevent livelock and deadlock (Grimley & Monroe, 1999; Van Aart, 2005). In Sudeikat *et al*. (2004) and Flores-Mendez (1999) various approaches are described to *organize* multi-agent societies for example in groups, teams, local areas or cooperation domain servers. The organizational structures specify the *relationships* among agents and agent types. The relationships consist of both inheritance relations and relations among agents based on their roles in the system. These o*rganizational structures* as well as *coordination strategies*, or problem-solving methods, can be used to allocate tasks to agents and control agents in multi-agent systems (Van Aart, 2005). Agents as a society thus operate under *global social laws and conventions* in their interactions with other entities in their environment.

A *coordination model* describes the interaction among agents and can be used to coordinate the interactions among agents and enact the social laws in the system by managing their organizational relationships or mutual dependencies. A coordination model is defined as a framework dealing with the communication and synchronization of a system of autonomous agents. To this end, a coordination model consists of three elements (Cabri *et al*., 2000; Zambonelli *et al*., 2001):

- The *coordinables*, or entities whose mutual interaction is governed by the model, namely the agents.
- The *coordination media,* consisting of the abstractions that enable agent interactions and the core around which the components of a coordinated system are organized, such as semaphores, monitors, channels, tuple spaces, blackboards, and so forth.
- The *coordination laws* that define the behaviour of the coordination media in response to interaction events. Coordination laws can be defined in terms of a *communication language* and a *coordination language*. The communication language is a syntax used to express and exchange data structures. The coordination language consists of a set of interaction primitives and their semantics.

The coordination media in the coordination model, such as the blackboards or tuple spaces employed, resides on the social layer. It arbitrates in *all* interactions between agents, and determines the effect of the interaction according to its embedded social laws. The coordination media can allow agents with heterogeneous communication laws or interaction protocols to interact. It can also constrain the behaviour of agents during interactions, thereby adding additional security (Zambonelli *et al*., 2001). If the organizational structures and coordination problem-solving methods are stored in libraries as patterns, agent behaviours can be modelled accordingly (Van Aart, 2005). Implementing a coordination model in the social layer therefore provides a situation where social tasks can be allocated and social laws enforced.

The mobility, communications and persistence aspects are placed inside the social layer, since all of them are subject to the social laws and conventions of the agent society. For example, there may be a global restriction on the length or number of negotiations between agents, which will be handled by the communications aspect, and the social layer should then ensure that this global law is obeyed. The mobility aspect may require that a third party be asked to serve as a code cache, and if certain servers in the agent society have been allocated to serve this purpose, the social layer should ensure that only these servers are used for this purpose. Similarly, there may be global restrictions on which resources may be used for storage purposes to allow persistence, which should be enforced by the social layer.

4.3.2.5 Mobility aspect in the social layer

Because of their higher level of autonomy, mobile agent systems require more than the readiness of the underlying network and the typical system requirements for mobile code to move these agents over the Internet. In fact, the mobility of many commercial mobile agent systems is restricted, being defined as an agent visiting a host, and then directly returning to its agency. This is largely due to the relative infancy of the field (moving agents vs. stationary agents) and the lack of programming standards to facilitate mobility.

Agent mobility can be implemented either as weak or strong mobility. In weak mobility, the agent's state is represented in data structures and migration is only allowed at specific points in the agent code. In strong mobility, the agent's state is captured at the underlying thread, which allows for migration at any point in the agent's execution (Tripathi *et al.*, 2001). With weak mobility, the agent will restart on the new host with its current data, while strong mobility allows the agent to continue execution from the point in its instructions where it was transferred (Horvat *et al.*, 2000). Most Java-based mobile agent systems support weak mobility, while systems not based on Java often provide strong mobility (Horvat *et al.*, 2000; Picco, 2001; Tripathi *et al.*, 2001). If the thread of control needs to be retained in an agent system supporting weak mobility, additional programming is required to save the execution state manually. In a system with strong mobility, migration is completely transparent to the migrated program, reducing programming effort as well as the size of the transported code (Picco, 2001). However, since moving to another host may require a mobile agent to perceive this new environment to determine for example what resources are available, restarting the agent, namely using weak migration, is considered more appropriate than strong migration (Hermann & Zapf, 2000).

Systems with weak mobility vary in the way agents request to move. Some use a simple 'go' statement causing the agent code and data to be moved to a new host, where a predetermined procedure or method is invoked. Others associate an itinerary listing the succession of hosts to visit with each agent and the method to invoke on each host. Some systems invoke the same method at each stop, while others allow the agent to specify different methods for each stop (Shih, 2001; Tripathi *et al.*, 2001).

Mobility often necessitates the transport of agent classes. Class transfers are usually approached in two ways (Gray *et al.*, 2002; Milojicic, 1999; Tripathi *et al.*, 2001):

1. all classes required by the agent code can be transported as part of the agent transfer protocol, or
2. classes can be obtained on demand from a designated code-base server during execution.

The purpose and circumstances under which the application will be used will determine which approach is followed.

D'Agents uses the first approach, which actually makes agent transfer heavyweight, since more classes than the agent actually needs may be transferred. Nevertheless, it has the advantage that the agent needs no further communication with its previous host for code transfer (Gray *et al.*, 2002). SMART uses the second approach, and allows the agent to attach new code and data at runtime through the process of dynamic aggregation. This reduces the amount of code transferred with the agent. It also reduces bandwidth requirements and speeds up the packing process before transmitting an agent to another host (Wong *et al.*, 2001). It slows down agent execution (Tripathi *et al.*, 2001) though and is not suitable for disconnected operations. An alternative approach is sometimes used where a third party could be asked to serve as code cache to allow the sending host to disconnect, or code could be cached at each host (Gray *et al.*, 2002). A number of variations on these approaches are discussed by Gray *et al.* (2002), Miljojic *et al.* (1999) and Triphathi *et al.* (2001).

4.3.2.6 Communication aspect in the social layer

Two of the critical characteristics of mobile agents are their capabilities to collaborate and to share information. Without interoperability it will be impossible to realise the potential capabilities and benefits of mobile agent technology. Mobile agent systems are interoperable if a mobile agent from one system can migrate to the second system, the agent can interact and communicate with other agents (local or even remote agents), the agent can leave this system and it can resume its execution on the next interoperable system (Pinsdorf & Roth, 2002). Interoperability aspects are often impeded by complexity and security concerns. Current trends include open agent systems, which are heterogeneous by nature, and thus may include more than one language. In the light of these trends, MASIF's claim that interoperability issues can be restricted to mobile agents written in the same language, as all mobile agent systems will be written in Java in the near future, does not hold. Furthermore, the behaviour expected from the mobile agent often dictates the source programming language, since different languages have varying domains of applicability (Dale, 1997). Ideally a mobile agent system should support several languages to accommodate different application needs (Gray *et al.*, 2002; Kotz *et al.*, 2002).

Mobile agents need to communicate with one another and sometimes also with other non-agent services and resources. Gray *et al.* (2002) point to four important communication issues to be included in the design of a mobile agent, namely

- how to identify another party with which to communicate;
- the required communication format;
- how to pass data from one party to the other (messaging); and
- how to maintain communication with a moving agent.

Since identifying an agent or a communicating party can be treated as inherent to each of these issues, only the last three of these communication issues are discussed with referrals as to how the identification issue has a bearing on each of them.

Innate to agent communication are the issues concerning the *required communication format*. Because mobile agents operate in a heterogeneous environment, interoperability cannot be achieved if there is no standardised means through which agents can understand and communicate with their functional environment. An ACL provides the capacity to integrate disparate sources of information (Finin, Labrou & Peng, 1998). An ACL is usually composed of an inner context language with a common vocabulary with ontology, an outer communication language and message passing mechanisms. Java and Tcl/Tk are commonly used as the inner context language. An ACL must support three important communication aspects, namely

- how to translate between different ACLs;
- how to guarantee the semantics of concepts, objects and relationships; and
- how to share the intended messages to be communicated.

There are many examples of standardisation efforts with regard to an ACL. Two examples are mentioned briefly. The *Knowledge Sharing Effort* (Genesereth & Finin, 1992; Neches, Fikes, Finin, Gruber, Patil, Senator & Swartout, 1991) developed three languages to address the abovementioned ACL issues: the *Knowledge Interchange Format (KIF), Ontolingua* and the *Knowledge Query and Manipulation Language (KQML)*. KIF was specifically designed to address the translation aspects of an ACL. Ontolingua addresses the semantics issue. KQML addresses the issue of sharing messages in inter-agent communication. In another example, one of the primary activities of the FIPA standard is to design a standard modelling language to support agent software engineering and to guarantee a high quality development process to construct multi-agent systems (Calisti, 2003).

Messaging plays an important role in inter-agent and agent-host communication. Aridor and Oshima (1998) discuss two basic methods for the delivery of messages:

- *Locate-and-Transfer*, where the agent is located with the aid of the naming services and then the message is transferred directly to it, namely two distinct phases exist during message delivery, and
- *Forwarding*, where locating the receiving agent and delivering the message occurs in a single phase.

The first may be problematic, since the agent may already have migrated when the second stage of transfer takes place. The latter may be more efficient for smaller messages, while a large message can be delivered more efficiently through the former method.

Communication maintenance approaches to address agent communication issues can be divided into three categories, namely synchronous communication (such as offered by TCP), asynchronous communication (IP, RMI or RPC) and indirect communication (event-notification and shared group/meeting objects) (Horvat *et al*., 2000; Tripathi *et al*., 2001). Synchronous communication can be subdivided into (1) communication where messages containing strings of arbitrary data are passed and (2) communication where messages containing serialised objects are passed (Gray *et al*., 2002). In all categories agents need to know each other's names to establish or maintain communication.

A demonstration of why it is important to distinguish between the different communication categories in mobile agent systems is offered by the following example. In synchronous communication, migration mechanisms must be included to cater for a participating agent since migration of the agent will disrupt the communication session. This will not happen in asynchronous communication, for if the agent is transferred, an exception will be thrown, and the initiating agent can simply re-execute the lookup and binding protocol to re-establish communication with the other agent at its new location (Tripathi *et al.*, 2001).

Most mobile agent systems use more than one of these mechanisms, as well as broadcasting/multicasting when sending the same message to multiple receivers (Horvat *et al.*, 2000). D'Agents, for example, uses all three by keeping a simple directory service, implemented as a collection of stationary cooperating agents (Gray *et al.*, 2002). Telescript agents (White, 1994) typically 'meet' other local agents and then invoke methods on objects in the other agents (example of asynchronous communication), but these agents can also communicate by sending events (an example of indirect communication).

Maintaining security during communication, especially while providing remote communication to visiting agents, is an important factor to consider during the design of a mobile agent system. A visiting agent could copy or transfer unauthorized information, gain access to protected resources, or launch denial of service attacks. Therefore encrypted and authenticated inter-agent communication needs to be supported (Tripathi *et al.*, 2001). Authentication is addressed on the *host security* and *agent security* layers.

4.3.2.7 Persistence aspect in the social layer

The concept of an agent migrating between hosts implies that the user is not reliant on the system launching the agent and is not (or should not be) affected if a host fails, therefore persistence is built into the notion of mobile agents (Harrison *et al.*, 1995). However, explicit persistence mechanisms should be integrated into a mobile agent system, to avoid loops and interminable waiting upon agents to return to their agent system. Furthermore, since mobile agents do not need to maintain a permanent connection, their state is centralised within themselves, implying that an improved fault tolerance can be expected from this technology. Nevertheless, specific mechanisms have to be embedded into mobile agent systems to handle fault situations, such as breakdown of connections or hosts, destruction of the agent, or network errors causing the agent to get lost.

While most systems offer little support for failure detection and recovery (Picco, 2001; Tripathi *et al.*, 2001), a number of systems provide some form of persistence and fault-tolerance for mobile agents by means of a *checkpoint-restore mechanism* to restart agents (Gray *et al.*, 2002; Horvat *et al.*, 2000; Picco, 2001). According to this mechanism the agent's state information is checked before and after execution on a host server, and when the server is restarted, a recovery process restarts any agents left on the server at last shutdown (Horvat *et al.*, 2000). The Tacoma architecture (Johansen, Renesse & Schneider, 1995) achieves persistence by allowing agents to create folders inside cabinets (storage space on disk) on boot-up. Tacoma then examines the 'system' cabinet and launches new agents for all the folders detected there. Other persistence mechanisms are discussed in Triphati *et al.* (2001), Feridun and Krause (2001) and Vogler *et al.* (1997).

4.3.2.8 Mobile agent management services layer

Serialization, deserialization and conversions, where necessary, are done on the *mobile agent management services* layer of the proposed architectural model. In essence *object serialization* in mobile agent technology is the ability to read and write objects to byte streams, commonly for saving session state information and for sending objects (agents and their states) over the network.

Serialization adds lightweight persistence as well as limited security to the agent as it protects private and transient data. Although object serialization in general does not contain any encryption/decryption algorithms in itself, it writes to and reads from streams, so it is possible for it to be coupled with any available encryption technology.

Java includes serialization through an API that allows the serialised data of an object to be specified independently of the fields of the class. Those serialised data fields can be written to and read from the stream using the existing protocol to ensure compatibility with the default writing and reading mechanisms. Aglet hosts use the standard *Java ObjectSerialization* mechanism to export the agent state as well as the aglet itself to a stream of bytes. However, the state of the execution stacks and program counters of the threads owned by the aglet are not serialised, which implies that when an aglet is dispatched, cloned, or deactivated, any relevant state sitting on any stack of a running aglet, as well as the current program counter for any thread, is lost (Venners, 1997). Grasshopper uses the MASIF serialization/deserialization process also based on the Java language. Examples of other methods of serialization (supporting other languages) are discussed in the Waterken Web (*Object Serialization Specification*, 2003), Le Goff *et al.* (Le Goff, Stockinger, Willers, McClatchey, Kovacs, Mar & Hassan, 2001) and Procopio (2002).

4.3.2.9 Agent security

A malicious hosting node can launch several types of security attacks on a mobile agent and divert its intended execution towards a malicious goal or alter its data or other information in order to benefit from the agent's mission (Sander & Tschudin, 1998). Bierman and Cloete (2002) describe four classes of security threats being imposed on mobile agents by malicious hosts, namely

- *integrity attacks*, which include integrity interference and information modification;
- *availability refusals*, which include denial of service, delay of service and transmission refusal;
- *confidentiality attacks* including eavesdropping, theft and reverse engineering, and
- *authentication risks*, which include masquerading and cloning.

In the same paper, three types of counter-measures were presented to address these classes of problems. The first type of counter-measure refers to *trust-based computing* where a trusted network environment is created in which a mobile agent roams freely and fearlessly without being threatened by a possible malicious host. A trusted environment can be achieved through tamper-resistant hardware, an agreement between specific hosts, and inclusion of detection objects in the mobile code. However, a trusted environment also goes against the notion of mobility, thus having access to unlimited Internet information. A second type of counter-measure that can be considered includes methods of recording and tracking that make use of the itinerary information of a mobile agent, either by manipulating the migration history or by keeping it hidden. Examples of specific recording and tracking methods include among others, itinerary recording and tracking mechanisms (Roth, 1998; Schneider, 1997), and server replication (Minsky, Van Renesse, Sheider & Stoller, 1996). The third type of counter-measure includes cryptographic techniques that utilise encryption/decryption algorithms, private and public keys, digital signatures, digital timestamps, and hash functions to address different threat aspects. Examples include cryptographic tracing (Vigna, 1997), encoding with encrypted functions (Sander & Tschudin, 1998), partial result encapsulation (Jansen, 2000), and others.

Many of the abovementioned methods are merely theoretical proposals at this stage and have not yet been implemented, owing to their complexity. As the malicious host problem is not simple, much research is currently being conducted to provide better detection and prevention mechanisms.

4.3.2.10 Naming server layer

A global *naming scheme* managing agent names is important for identifying, controlling and locating agents (Milojicic *et al.*, 1998). Moreover, a mobile agent system requires a *name service* to locate resources, specify agent servers for migration and also to establish inter-agent communication. A naming scheme defines a namespace, which may be hierarchically structured. A name service maps a resource name to its physical address and could also indicate the type of resource as well as the protocol to access it (Tripathi *et al.*, 2001). Since communicating agents need to agree in advance on the naming server, an architecture where agent servers share a default naming server simplifies the registration system. The *naming server* layer therefore implements a global naming scheme and name service.

MASIF uses an agent's name (as assigned by its authority), its identity (a unique value within the scope of the authority) and the agent's system type (for example Aglet) to form a globally unique name for each agent. As described earlier, MASIF uses four basic techniques to find an agent, including brute force, logging, agent registration and agent advertisement. Other systems also use these, a hybrid of these, and additional methods to locate agents. For example, Aridor and Oshima (1998) use the brute force search in parallel or in sequence, as well as a database for agent registration as a predefined directory server used to register, unregister or locate agents. Other agents can then use this directory to find the agent.

4.3.2.11 Host security

A host is faced with two potential threats from mobile agents, namely a *malicious agent*, which might be a virus or Trojan horse vandalising the host, or a *benign agent* that might simply abuse a host's local resources. In an uncontrolled environment, mobile agents can potentially run indefinitely and consume the system level resources such as files, disk storage, I/O devices, etc., in their execution environment. Therefore resource consumption, such as CPU time, disk storage, number of threads, number of windows, network bandwidth and the like, should be limited and access control policies that define accessibility to these resources must be put in place (Feridun & Krause, 2001; Gray *et al.*, 2002; Picco, 2001; Tripathi *et al.*, 2001).

In both instances, three basic types of counter-measures can be applied, namely *authentication*, *verification* and *authorisation*. Authentication ensures that the agent comes from a trustworthy site. Verification checks the agent code to ensure that it does not attempt to corrupt the execution environment by performing prohibited actions. Authorisation checks the runtime actions of the agent to ensure it does not exceed allowances or access permissions. Verification mechanisms address the malicious agent threat. Protection mechanisms against a potentially malicious host can of course prevent such verification procedures from being implemented. This means that the verification mechanisms can only be confined to the execution environment and hence only manage the agent during execution to ensure that the agent does not try to corrupt the execution environment (Dale, 1997).

The XML-based Security Assertion Markup Language (SAML) *(Assertions and Protocol for the OASIS Security Assertion Markup Language 3 (SAML) V1.1.4,* 2003*)* is a recent standardisation effort to address the *authentication* and *authorisation* concerns. One of the major design goals of SAML is single sign-on, which implies the ability of a user to authenticate in one domain and use resources in other domains without re-authenticating. Although the OASIS standard is primarily aimed at Web services that allow the exchange of authentication and authorization information among business partners, it is of particular importance for mobile agents as they engage and interact with resources and other agents in various domains. With it, a security administrator can express advanced security requirements, such as time- or event-based restrictions.

4.3.2.12 Network layer

The network layer is responsible for the final encoding of the encrypted serialised agent object so that the underlying network can transport it to its next host. Although there are several ways to perform this encoding, the IBM Aglet's workbench team proposed an ATP as a mobile agent standard to be adopted for transporting mobile agents. ATP is a platform-independent, application-level protocol for distributed agent-based systems for the purpose of transferring mobile agents between networked computers (Lange & Oshima, 1998). Some other mobile agent systems simply use the underlying HTTP or TCP/IP protocols as transport protocol, for example Aridor and Oshima (1998) and D'Agents (Gray *et al.*, 2002).

Security mechanisms can also be included in the agent's transport protocols. For example, protocols such as Secure Socket Layer (SSL) and Transport Layer Security (TLS) (*Transport Layer Security*), although a bit heavyweight, can be used for securing transmission of data between two hosts. Alternatively, the Key Exchange Protocol (KEP) (*Internet Key Exchange Security Protocol*) offers a lightweight transport security mechanism, which suits the notion of small transferable objects better.

Layer	Purpose	Significance
Authority API	API that facilitates interaction with and management of its mobile agents both locally and at remote hosts. Includes tasks such as agent creation, communication and destruction.	Without the authority API, the mobile agent system is too complex for commercial acceptability. Depending on the client, it is most likely that the user is not a programming expert and therefore would either not be able to or would not want to interact with the programming environment.
Agency	Provides an environment where agents can be created, suspended, activated and destroyed.	This layer allows an agent system to control both its own agents and visiting agents from *another* agent system (Milojicic *et al*., 1998).
Execution layer	Provides constrained execution environments (places) where agents can execute and meet other agents.	The constrained execution environment provides all the services necessary to execute a mobile agent, but limits their actions. It allows multiple agents to live and execute on the same server without interfering with each other and allows agents to communicate with each other. To a certain extent it thus protects both the host and the agents against each other.
Social layer	Implements organizational structures and coordination strategies to allocate social tasks and enforce global laws and conventions in the society of agents.	Without coordinating the agents' actions, they may perform tasks that are counterproductive to each other, leading to livelock[11] and/or deadlock in the system (Grimley & Monroe, 1999). Agents' actions also need to be coordinated to allow them to achieve both their individual and common goals.
Mobility aspect	Implements mechanisms to ensure mobility such as providing support for transport of data and agent classes.	The agent code may require class transfers when an agent is created remotely or transferred, since the `Agent` class is needed to instantiate the agent. If it does not exist on the host, it must be transferred from the source. Other classes used during agent execution, if they are not available at the destination, must also be transferred from the source or from a server (Milojicic *et al.*, 1998). Weak mobility may also require additional mechanisms to transfer data. Without the transfer of the required classes or data, the mobile agent will be unable to restart and/or continue its execution.
Communication aspect	Implements messaging, which includes broadcasting, and maintains communication with moving agents.	Without communication, mobile agents will be unable to collaborate and share information, and thus, to a large extent, unable to achieve their goals.

[11] Livelock occurs when agents act continuously, but make no progress towards the overall goal (Grimley & Monroe, 1999)

Layer	**Purpose**	**Significance**
Persistence aspect	Implements fault-tolerance and persistence mechanisms.	Various situations, such as breakdown of connections or hosts, destruction of the agent or network errors causing the agent to be lost, can prevent an agent from migrating successfully (Horvat *et al.*, 2000; Vogler *et al.*, 1997). Fault-tolerance and persistence mechanisms provide mechanisms for local recovery. Without these, loops may arise or interminable waiting upon agents to return, may occur.
Mobile agent management services	Serialises the agent before transmission and deserialises it on arrival at the remote host.	Serialization allows saving the state of the mobile agent and writing it to a byte stream in order to transfer it over the network. On arrival at the remote host, deserialization allows reconstruction of the mobile agent and enables it to continue its execution. Without it, the execution state of the agent would be lost.
Agent security	Protects the agent against malicious hosts and during transfer. Encrypts agent before migration and decrypts new agents before execution.	Without agent security, malicious hosts and other agents could modify an agent's code, state or data so that the agent does not achieve its goal, or achieves a different goal than the one intended. Encryption protects an agent's code, state and data during transmission against other agents and malicious hosts under way.
Naming server layer	Implements a global naming scheme to identify, control and locate agents, and a name service to locate resources, specify agent servers for migration and establish inter-agent communication. Names and registers newly created agents for future reference. Registers new agents that arrive at the host. Determines location of hosts to be visited.	Without a globally unique name for each agent, it would be impossible to identify, control and locate a specific agent in a mobile agent system. This in turn implies that it would be impossible to authenticate and authorize the agent. Authentication ensures that the agent is trustworthy, and authorization checks the runtime actions of the agent to ensure that it does not exceed its access permissions. Both of these are linked to the agent's identity. Agent identification is also important to enable communication with other agents, as well as with the environment. This layer thus makes it possible to implement host security, to trace agents and transfer them to their remote hosts, and enables communication.

Layer	Purpose	Significance
Host security	Provides credentials for access to other hosts to the mobile agent before migration. Authenticates agents received from remote hosts, verifies and authorises their code to protect the host against malicious agents. Enforces access control policies to protect the host against abuse of its local resources.	Without host security, malicious agents can access and/or modify data and resources available at the server, or infect the server. This can lead to denial of services, including disrupting execution of other agents and opposing services to other agents. Access to the host can be abused, for example by formatting the hard disk and harming the server or other agents (Dale, 1997; Harrison *et al*., 1995; Lingnau *et al*., 1995). This layer therefore protects the host environment.
Network	Encodes the agent in a suitable transport protocol for transport through the network.	This layer represents the hardware and software protocols necessary to enable mobile agents to migrate. Without it, the mobile agent would not be able to migrate.

Table 4.3 A summary of the different layers in the proposed architectural model

4.4 Significance of the proposed model

The proposed generic mobile agent system architectural model provides a comprehensive view of the architectural components required to implement a mobile agent system in agent orientation. This includes the social characteristics required by multi-agent systems, which are indicated by the social layer.

Though most of the platforms used to identify the essential architectural components follow a layered approach, none of them includes all the layers in the proposed architectural model. Grasshopper is the most comprehensive of the platforms described. Its architecture though contains two aspects, that make it difficult for a novice to comprehend. It consists of two layers, which appear to be mapped upon each other. Within these two layers, components are layered within several other components. The architectural components in the proposed architectural model, in contrast, represent the essential features of a mobile agent system in a simplified, general but clear way. A clear understanding of the architectural components that are involved makes it much easier for a novice to form a broad overview of the important system elements during design. It also enables programmers to identify and include different mobile agent characteristics, with maximum reuse of available technologies and architectures. No other model with these features was found in the literature.

The layered approach isolates the individual architectural components that each form an essential part of the mobile agent system. This enables developers to focus on the tasks that apply to a specific layer during its development, since each layer forms a functional unit. A layered model also allows the independent development of layers while implementing each layer separately and also supports program maintenance, debugging and upgrading.

Furthermore, the social layer, in representing the social character of a multi-agent system, reminds the novice that a paradigm shift is necessary to implement mobile agent systems. As pointed out earlier, to realize the full potential of mobile agent systems, they should be implemented according to the proper principles in agent orientation.

In the discussion of each component in the proposed architectural model, issues to take into account in order to enhance the design and development of mobile agent systems, are pointed out. This provides further guidelines for building commercially acceptable mobile agent systems.

The proposed architectural model thus provides a constructive tool to both programmers and researchers who enter the field for the first time. The architectural model could act as a first step on the way to establishing a standard that can be used during the design and development of mobile agent systems. It is also significant because it provides a point of reference for researchers and developers to evaluate the capabilities of mobile agent systems created by commercially available tool kits.

4.5 Proposed model application and comparison

The significance of the proposed architectural model becomes apparent when one compares it to a real-life example. This example is provided by Project TUTA (Tshwane University of Technology Agents) (Mentz & Bierman, 2004). Project TUTA was formulated by the Tshwane University of Technology (TUT) and is based on a dissertation submitted by Mentz (2004). For this project, three programming groups each received design requirements for a research-oriented mobile agent system. One group consisted of five students and the other two groups of four students each. The assignment was to produce a working mobile agent system in which at least one agent migrates between at least two different execution environments. The mobile agent had to migrate to the various destinations stored in its itinerary, execute code at each of those nodes and store the results in its memory. Implementations were done in C# in the .Net environment.

The initial description of the system suggested that each node store air flight information and that the mobile agent migrates to each of these nodes to collect information such as the cheapest flight to a destination. However, each group could also choose its own goal. Eventually all the groups implemented different applications. One group searched for a book in a database, another attempted to coordinate a meeting between different parties, and the third group simply tried to achieve mobility.

Project TUTA design requirements consisted of a descriptive model describing the relationships between mobile agent system elements. It contained four layers. The bottom layer consisted of the underlying network technology and was accepted as existing and appropriate for the system. The second layer was provided by the operating system, which acted as interface between the underlying network and resources for the third layer, the execution environment. The topmost layer consisted of the mobile agent itself.

The execution environment contained five managers:

- The *migration manager* was responsible for transferring the mobile agent to the desired node. This included preparation for migration, communication with the destination node, transfer of the mobile agent and completion of migration.
- The *security manager* ensured the safety of the mobile agent, the host visited by the mobile agent and the resources of the host.
- The *mobile agent manager* controlled the mobile agent life cycle in cooperation with the *resource manager* and the *communications manager*. It also contained a GUI to show relevant information about mobile agents and their activities.
- The *resource manager* provided mechanisms for access to resources other than basic execution resources, such as databases.
- The *communications manager* provided communication channels to other mobile agents or entities outside the system.

The Project TUTA architecture is depicted in Figure 4.6.

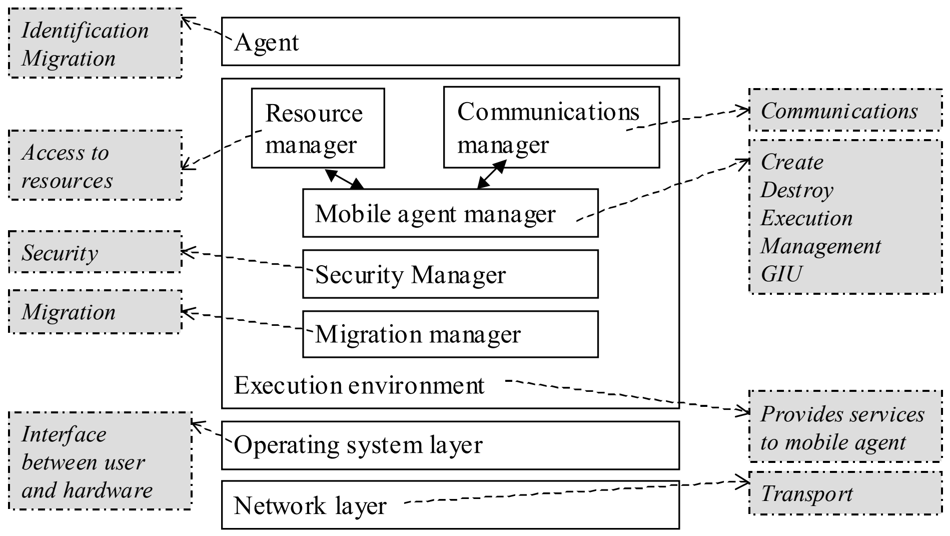


Figure 4.6 Project TUTA architecture

Of the three groups in Project TUTA, only one group followed the design architecture given to it. This was also the only group to produce a functional mobile agent system. In this case the purpose of the system was that the agent would visit a database containing information about books and then report back whether or not a title had been found that matched the given information.

To illustrate the importance of an architectural design, the rest of this discussion focuses on the design of the programming group whose mobile agent system attempted to coordinate a meeting between different parties. While their demonstration of the system proved that the mobile agent was indeed able to reach other hosts, it was not able to coordinate the meeting. One of the complaints of this programming group was that the system became too big and its clumsiness prohibited them from including all the functionality. Their failure demonstrates a problem at design level rather than development level. It emphasizes both the importance of analysis and the subsequent design processes, as well as following an iterative cycle in this process when implementing a system.

If the programmers used a modular design based on different architectural components, they would most probably have incorporated most of the necessary elements into their design without being overwhelmed by the size of the system. In this case slipshod communication infrastructure or persistence design most probably introduced most of the problems and programming failures.

Missing layers in the architectural design could cause novices to omit or neglect important aspects when developing a mobile agent system. For example, it is possible that, without a global naming scheme and a naming service, when more than one mobile agent system is deployed on the same network in Project TUTA, confusion may ensue if the previous demonstration(s) had not been deleted.

4.6 Conclusion

This chapter proposed a generic mobile agent system architectural model that can guide novice mobile agent programmers when developing a mobile agent system. The model provides a comprehensive overview of the different architectural components for a mobile agent system in the agent orientation paradigm. The proposed architectural model includes more layers than any of the platforms used as a basis to identify components, and thus addresses more aspects, enabling it to provide more assistance to novices. The Project TUTA example points out the importance of architectural design. This shows that the proposed model can certainly act as a tool to assist novice mobile agent programmers by ensuring that all the necessary elements are included during system development.

In Chapter 5 a knowledge base to encapsulate the concepts presented in Chapter 2 to 4 is discussed.

CHAPTER 5

5 KNOWLEDGE BASE FOR NOVICE MOBILE AGENT PROGRAMMERS

5.1 Introduction

This chapter describes one possible visualization of the knowledge base that was developed to assist novice mobile agent programmers in order to help them to develop commercially acceptable mobile agent systems. To this end the chapter contains seven sections. Section 1 is this introduction, and section 2 discusses the importance of the knowledge base for novice mobile agent programmers. Section 3 describes the design and development of the knowledge base environment, while section 4 describes the implementation of the knowledge base environment. Section 5 provides some examples of the interaction between the novice mobile agent programmers and the knowledge base. Section 6 reports on the testing of the knowledge base, while section 7 concludes the chapter.

Knowledge management is an evolving discipline that seeks to make the best use of available knowledge, creating new knowledge and increasing awareness and understanding in the process. For this purpose, a separation may be made between concepts of data, information, tacit knowledge and explicit knowledge. Data is considered to be factual, raw material without contextual meaning. Information is data that has been refined into a structural form, for example a database. Explicit knowledge relates to "knowing about" and is storable and transferable, for example manuals, specialized databases or collections of programming libraries. Tacit knowledge relates to "knowing how". It is not directly transferable between individuals, but through application, practice and social interaction (*Knowledge Management*, 2005).

The intended knowledge base incorporates all these concepts in order to construct tacit knowledge that is transferable through application of the presented knowledge units. A systematic approach is followed to implement the knowledge units described in previous chapters into a usable tool for the novice mobile agent programmer.

As mentioned earlier, a knowledge base is

'a special kind of database for knowledge management. It is the base for the collection of knowledge. Normally, the knowledge base consists of explicit knowledge of an organization, including

troubleshooting, articles, white papers, user manuals and others. A knowledge base should have a carefully designed classification structure, content format and search engine.'(Knowledge Base, 2005*)*

From this definition, it can be seen that the focus of this research effort is not on the architectural design of the knowledge base. An elaborate description of the design elements will not contribute to the programming experience of the novice mobile agent programmer. Instead, it would deviate the focus from the objectives of this research that were established in the first chapter. Rather, for the novice mobile agent programmer, the knowledge base forms a programming aid or a tool. Thus, the design, development and general deployment issues of the tool are not of any concern to the programmer, nor will these contribute to his or her understanding of the mobile agent programming environment. For this reason (as stated in the scope of the dissertation), the discussions in this chapter are limited to issues that pertain to the objectives of this dissertation. However, for the sake of completeness, a brief overview of some of the structural issues with regard to the knowledge base is provided in the section 5.3.

5.2 Importance of the knowledge base for the novice mobile agent programmer

Developing agent systems requires specialized skills and knowledge in various areas. For this reason the use of agent construction toolkits are often recommended in order to implement agent systems. Even so, novice mobile agent system programmers need to understand and be aware of a substantial number of concepts, both at the level of implementing individual agents and agent societies, and at the level of constructing the environment in which mobile agents operate, namely the agent systems, in order to develop commercially viable systems. Furthermore, novices attempting to program mobile agents often have difficulty in distinguishing between the concepts applicable to the agents themselves and those applicable to the agent execution environment, since there is some overlap, as well as in applying the concepts in the correct context. Both literature (Rothermel & Schwehm, 1998) and practical experience show that at least twelve months of (intensive) literature study are necessary to gain sufficient knowledge and insight to apply the acquired learning.

To facilitate and simplify this process, the literature study in Chapters 2 and 3 provides a survey of the programming milieu for mobile agents, while Chapter 4 provides guidelines for implementing mobile agent systems with a generic mobile agent system architecture. The knowledge base offers a summary of the knowledge gained during the previous chapters. It can be used as a tool both to refresh concepts while novice mobile agent programmers orientate themselves to this environment, and to serve as a reference during the process of designing and developing a mobile agent system. The examples, although fairly limited, are especially helpful.

The nature of the knowledge base is such that it can be used in conjunction with the accompanying dissertation or as a stand-alone reference. The references to related articles and websites provide guidance to users for independent further investigation, as well as links to a number of existing mobile agent systems.

5.3 Design and development of the proposed knowledge base environment

The knowledge base environment was designed to offer a summary of the knowledge gained during the previous chapters. The candidate did not do the actual programming thereof. To avoid being caught up in a separate systems analysis and design project, a rapid application development tool, namely Borland Delphi 7, was used to construct a knowledge management environment. An Access 2000 database is used to store the knowledge contained in the knowledge base. The system was tested on an HP compaq nc4010 notebook with an Intel® Pentium® M memory chip, a clock speed of 592 MHz, a 1.80GHz processor and 992 MB RAM; and an LG personal computer with an Intel® Pentium® 4 memory chip, a clock speed of 1.62 GHz, a 1.60GHz processor and 512 MB RAM. It also ran on a variety of other Windows-based machines.

This environment then communicates with a database that stores the relevant mobile agent knowledge units. The primary source of knowledge units is from the prior chapters in this dissertation. These chapters capture the relevant information pertaining to mobile agents within the context of constructing mobile agent systems. Figure 5.1 offers a graphical view of this process.

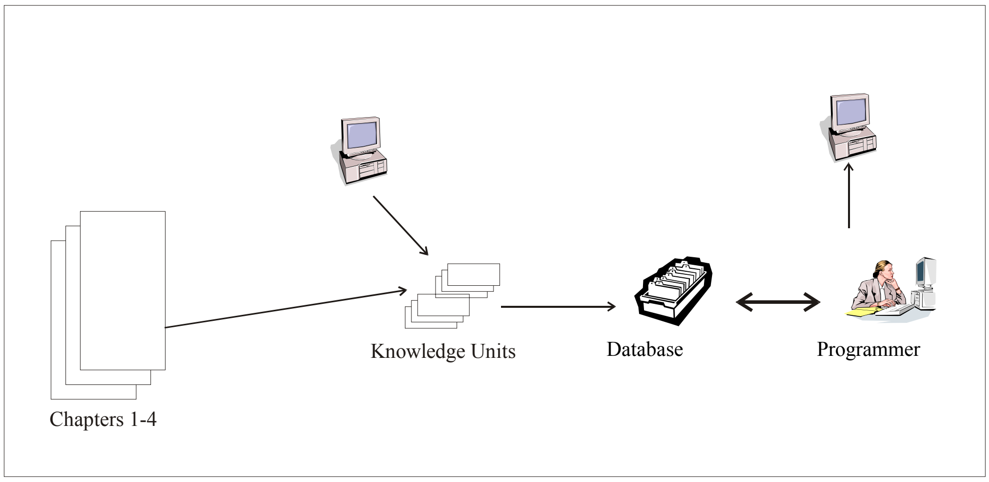


Figure 5.1 The essential interactivity between the gathered data and information to create explicit and tacit knowledge on the part of the novice mobile agent

The following structures were designed for each knowledge unit that had been identified as being relevant to the novice mobile agent programmer:

- *Topic* or theme of the knowledge unit
- Relevant *keywords*
- *Description* or clarification
- *Example* (if applicable)
- *Section reference* to refer the programmer to a point of reference in the dissertation, should the excerpt not cover his or her expectations sufficiently.
- Other *reference(s)* to related article(s) or websites.

A simple relational database consisting of a set of three tables with fixed columns, maintains the entire structure of the knowledge base. The main table *TopicTable* consists of four columns for the *Topic*, the *Description*, the *Example* and *Section*. Since a topic may have more than one keyword, as well as more than one reference, separate tables are used for the keywords and references in order to share information and minimize data redundancy. The *KeywordTable* consists of two columns, one for the *Topic* and one for a *Keyword*. Similarly, the *ReferenceTable* consists of two columns, one for the *Topic* and one for a *Reference*. Both the *KeywordTable* and the *ReferenceTable* are linked to the *TopicTable* with *Topic* as the common field. The three tables are depicted in Figure 5.2.

TopicTable

Topic	*Description*	*Example*	*Section*

KeywordTable

Topic	*Keyword*

ReferenceTable

Topic	*Reference*

Figure 5.2 Tables used in relational database used to implement knowledge base

Column values are commonly atomic[12] or are allowed to be NULL. For example, the knowledge unit "multi-agent system" will not contain sample code (as defined in the *Example* field) as this is irrelevant to the explanation as to what a multi-agent system is. In this case the *Example* field reverts to NULL. On the other hand, the concept "initiate mobile agent" requires the inclusion of sample code to illustrate the initiation of a mobile agent. In the latter case, the *Example* field will contain a description (as is typically defined in a MEMO field).

The following example further illustrates the implementation of the database design: Both the *Topic* and the *Keyword* fields allow for the creation of indexed files to present information to the user. This implies that the layout of the knowledge management environment is structured in such a way that the programmer can select from either a list of *topics* or a list of *keywords*. If a topic is selected, the knowledge management environment will, in its communication with the underlying database, match the selected input with the available topics. Once the topic is found, the knowledge management environment displays all relevant information related to the searched input. This includes the topic (which is also the indexed key field), a clarification of the topic and which section in the dissertation pertains to the topic, as well as sample code and reference(s) to related articles where appropriate. In

[12] Does not contain structured values or a collection of values

some instances sample code is not sensible. For example, if the searched term was “agency”, then the following information is appropriate:

TOPIC:	agency
DESCRIPTION:	Agency: Agency regards an agent as an autonomous software system acting in a continuous Perceive-Reason-Act cycle as part of a dynamic and open environment in order to achieve a goal. Weak agents are autonomous, have social ability, and are reactive and pro-active. Strong agents also have, in addition to the requirements for weak agency, mentalistic attitudes such as knowledge, belief, intention and obligation.
EXAMPLE:	Not applicable
SECTION:	2.2.1
REFERENCE(S):	Franklin, S. & Graesser, A. 1996. Is it an Agent, or just a Program? A taxonomy for Autonomous Agents. in *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag. Lange, D. B. 1998. Mobile Objects and Mobile Agents: the future of distributed computing? in E. Jul, (ed.) *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'98), LNCS*. vol. 1445. Springer-Verlag. Brussels, Belgium. 1-12. Lind, J. 2001. Issues in Agent-Oriented Software Engineering. in P. Ciancarini & M. J. Wooldridge, (eds.). *Agent-Oriented Software Engineering - Proceedings of 1st International Workshop AOSE-2000*. vol. 1957. Springer-Verlag. Berlin. 45-58.

From the information given in the *Description* field above, it becomes obvious that sample code, or another type of example, is not applicable in this case since the searched term explains a concept rather than a specific implementation detail. More examples follow in 5.5.

Each keyword is linked to a topic or topics. Since a particular keyword may be linked to more than one topic, when a keyword is chosen, a list of topics related to the keyword is displayed. Once a topic is chosen, the process proceeds in the same way as described in the preceding paragraphs.

A functional view of the software design is presented in Figure 5.3. It consists of a sequential system with a single control loop, which fetches and executes commands in sequence. The initial user interface and each of the three displays are objects. To display a list of available topics in the knowledge base, the column containing the topics is retrieved from the *TopicTable* and displayed. Similarly, to display a list of keywords, the list of keywords is retrieved from the *Keyword* column in the *KeywordTable* and displayed. SQL statements are used to match topics linked to the requested keyword in order to extract an abbreviated list of topics. The display of information relevant to a specific topic is also based on SQL statements.

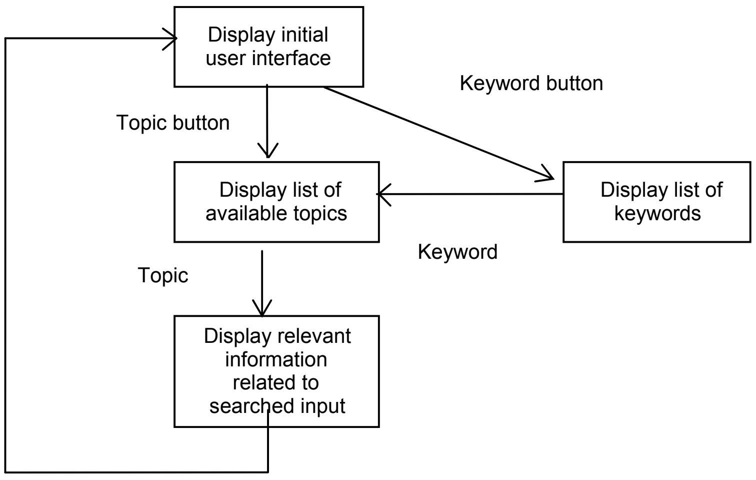
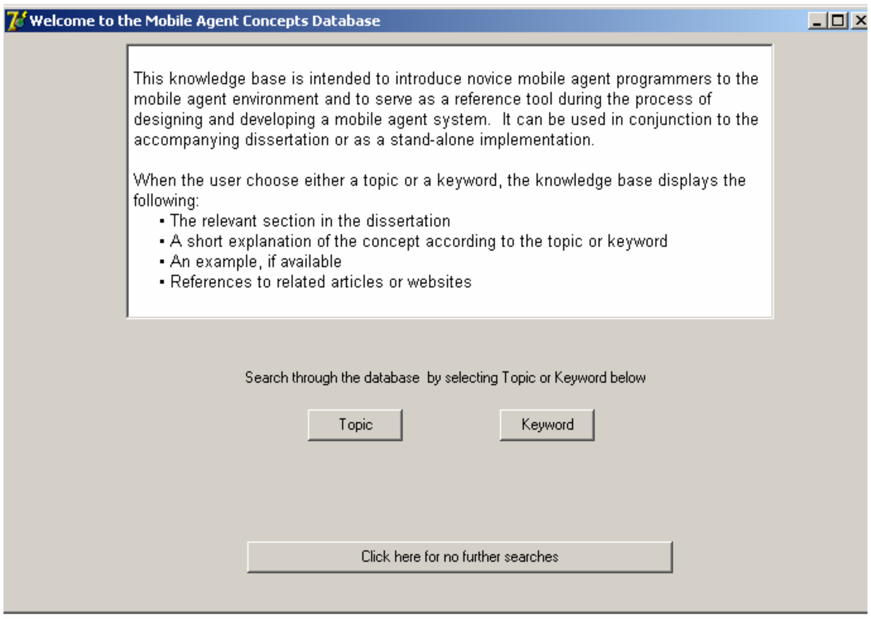

Display initial user interface
Keyword button
Topic button
Display list of available topics
Display list of keywords
Keyword
Topic
Display relevant information related to searched input

Figure 5.3 Software design of knowledge base

5.4 Implementation of the knowledge base environment

5.4.1 Functioning

Figure 5.4 illustrates the initial user interface when the knowledge base environment is activated.



This knowledge base is intended to introduce novice mobile agent programmers to the mobile agent environment and to serve as a reference tool during the process of designing and developing a mobile agent system. It can be used in conjunction to the accompanying dissertation or as a stand-alone implementation.

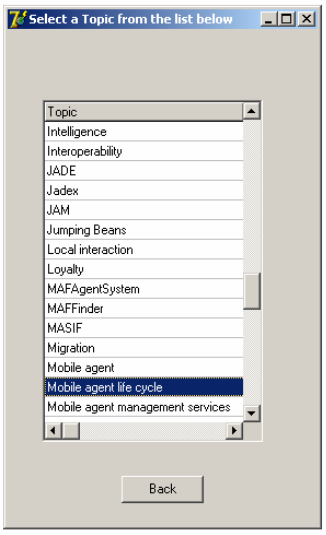
When the user choose either a topic or a keyword, the knowledge base displays the following:

- The relevant section in the dissertation
- A short explanation of the concept according to the topic or keyword
- An example, if available
- References to related articles or websites

Search through the database by selecting Topic or Keyword below

Figure 5.4 Entry to the knowledge base

As shown, the interface provides an introductory purpose statement and invites the user to make a choice between the *Topic* and the *Keyword* button to initiate his or her search. To avoid misspelled terms and idle searches, a complete list of all available topics or keywords will be displayed upon selection of either button. The user browses down through the individual items. Figure 5.5 illustrates an excerpt from the lists of topics when the *Topic* button is selected.



Topic
Intelligence
Interoperability
JADE
Jadex
JAM
Jumping Beans
Local interaction
Loyalty
MAFAgentSystem
MAFFinder
MASIF
Migration
Mobile agent
Mobile agent life cycle
Mobile agent management services

Figure 5.5 Entry to the knowledge base

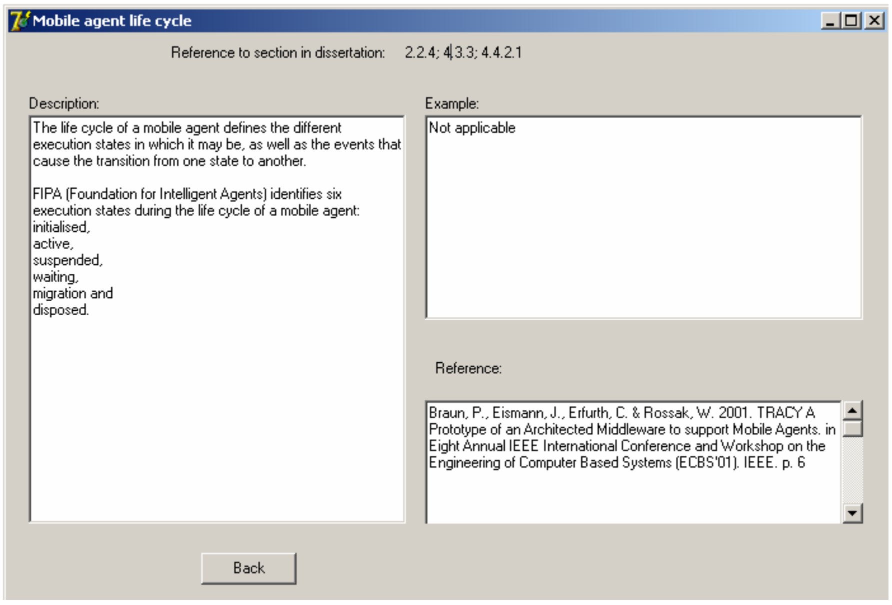
The list of available topics is displayed in a scrolling menu that allows the user to view all the available topics. Either the scroll bar or the down arrow on the keyboard may be used to scroll through the list of available keywords. Picking the *Back* button causes the user to return to the initial user interface. A topic is selected by pointing with the mouse. Selecting a topic causes the relevant information linked to the searched input to be displayed. This includes the following:

- The relevant section in the dissertation
- A short explanation of the concept according to the topic or keyword
- An example, if available
- References to related articles or websites.

As shown in Figure 5.5, the topic *Mobile agent life cycle* is selected. This causes the relevant information to be displayed, as shown in Figure 5.6.

The *Reference to section in dissertation* note at the top of the display indicates section(s) in the dissertation where the life cycle of a mobile agent is discussed or referred to. The clarification regarding the life cycle of a mobile agent is displayed in the *Description* text box. There is no code example associated with the life cycle of a mobile agent, and therefore the *Example* text box displays 'Not applicable'. References to articles containing further information on the life cycle of a mobile agent is displayed in the *Reference* text box. Note that when more than one reference is available, as in this case, a scroll bar to the right of the *Reference* text box allows scrolling to view all the references.

Selecting the *Back* button causes the user to return to the initial user interface (shown in Figure 5.4) where the user is once again invited to choose between the *Topic* or *Keyword* buttons to initiate a new search.


Mobile agent life cycle

Reference to section in dissertation: 2.2.4; 4.3.3; 4.4.2.1

Description:

The life cycle of a mobile agent defines the different execution states in which it may be, as well as the events that cause the transition from one state to another.

FIPA (Foundation for Intelligent Agents) identifies six execution states during the life cycle of a mobile agent:
initialised,
active,
suspended,
waiting,
migration and
disposed.

Example:

Not applicable

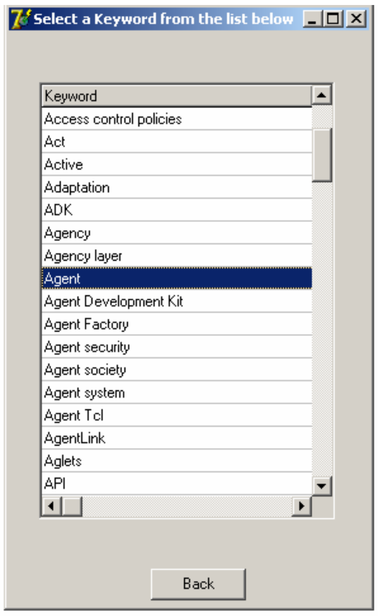
Reference:

Braun, P., Eismann, J., Erfurth, C. & Rossak, W. 2001. TRACY A Prototype of an Architected Middleware to support Mobile Agents. in Eight Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'01). IEEE. p. 6

Back

Figure 5.6 Information relevant to the topic *Mobile agent life cycle* is displayed

Selecting the *Keyword* button on the initial user interface (displayed in Figure 5.4) initiates the display of the list of keywords on a scrolling menu, as shown in Figure 5.7.

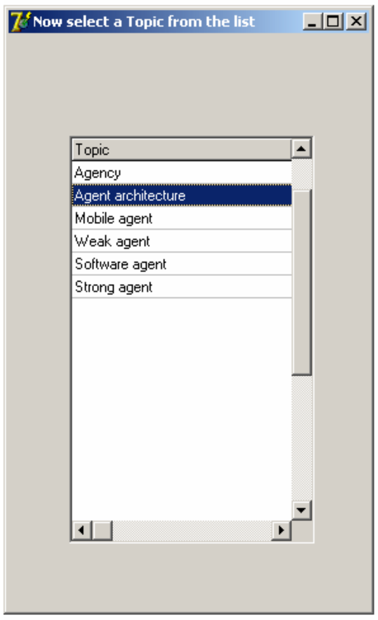

Select a Keyword from the list below

Keyword
Access control policies
Act
Active
Adaptation
ADK
Agency
Agency layer
Agent
Agent Development Kit
Agent Factory
Agent security
Agent society
Agent system
Agent Tcl
AgentLink
Aglets
API

Back

Figure 5.7 Extract from the list of available keywords

The scrolling menu allows the user to view all the available keywords. The user may use either the scroll bar or the down arrow on the keyboard to scroll through the list of available keywords. Similar to the display for the list of topics, selecting the *Back* button causes the user to return to the initial user interface. A keyword is selected by pointing with the mouse. Selecting a keyword causes an abbreviated list of topics consisting of those linked to the searched keyword to be displayed. Picking the keyword *Agent*, as shown in Figure 5.7, causes the abbreviated list of topics shown in Figure 5.8 to be displayed.



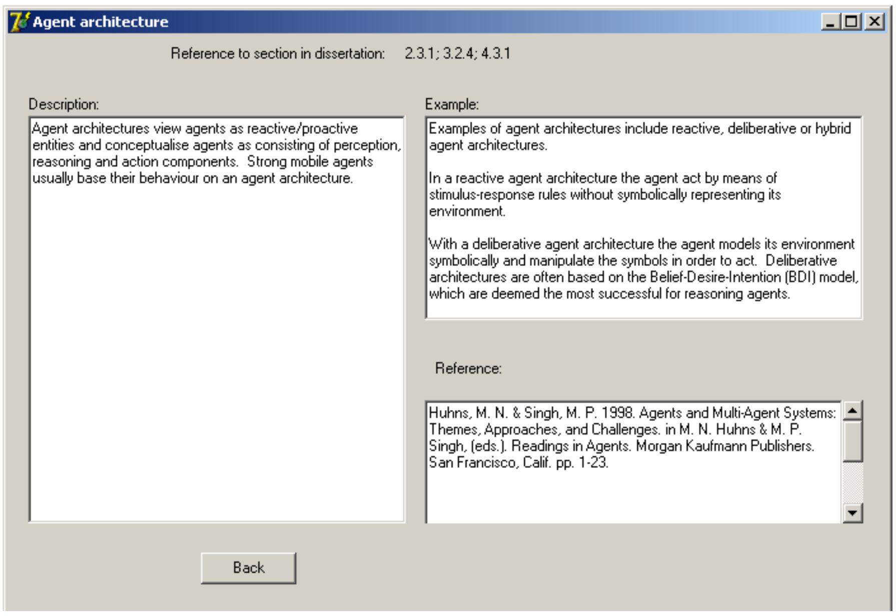
Topic
Agency
Agent architecture
Mobile agent
Weak agent
Software agent
Strong agent

Figure 5.8 List of topics linked to keyword *Agent*

Selecting a topic from the abbreviated list causes the relevant information linked to the searched input to be displayed. This includes the same set of information to be displayed as if the topic was picked from the original scrolling menu for topics, namely

- The relevant section in the dissertation
- A short explanation of the concept according to the topic or keyword
- An example, if available
- References to related articles or websites.

Picking the topic *Agent architecture* from the abbreviated list of topics shown in Figure 5.8, causes the relevant information to be displayed as shown in Figure 5.9. There are two reference papers containing more information on agent architecture and thus a scroll bar is provided in the *Reference* box.



Reference to section in dissertation: 2.3.1; 3.2.4; 4.3.1

Description:

Agent architectures view agents as reactive/proactive entities and conceptualise agents as consisting of perception, reasoning and action components. Strong mobile agents usually base their behaviour on an agent architecture.

Example:

Examples of agent architectures include reactive, deliberative or hybrid agent architectures.

In a reactive agent architecture the agent act by means of stimulus-response rules without symbolically representing its environment.

With a deliberative agent architecture the agent models its environment symbolically and manipulate the symbols in order to act. Deliberative architectures are often based on the Belief-Desire-Intention (BDI) model, which are deemed the most successful for reasoning agents.

Reference:

Huhns, M. N. & Singh, M. P. 1998. Agents and Multi-Agent Systems: Themes, Approaches, and Challenges. in M. N. Huhns & M. P. Singh, (eds.). Readings in Agents. Morgan Kaufmann Publishers. San Francisco, Calif. pp. 1-23.

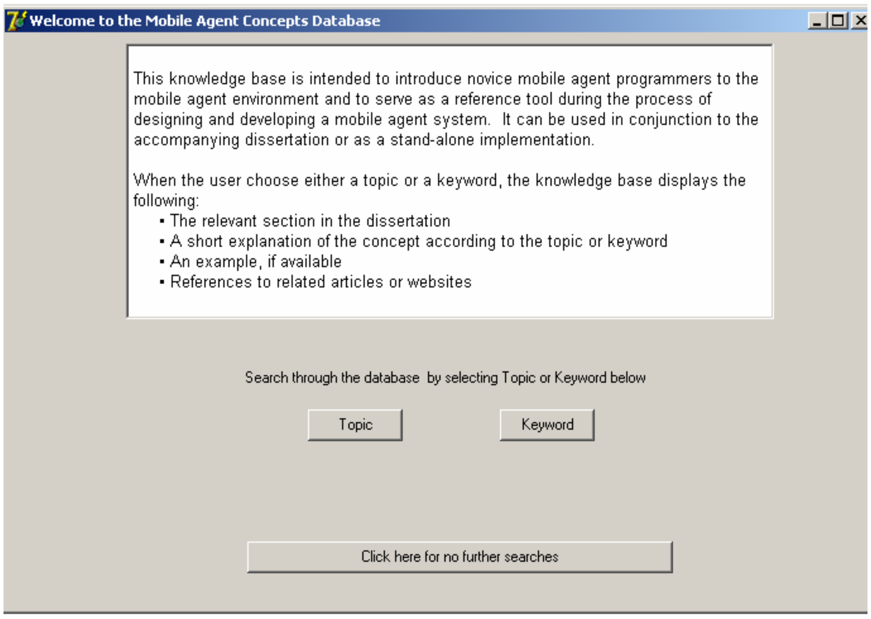
Figure 5.9 Information relevant to the topic *Agent architecture* displayed

Once again, the *Back* button causes a return to the initial user interface shown in Figure 5.4. Picking the *Click here for no further searches* button on the initial user interface (Figure 5.4) will close the knowledge base management environment.

5.5 Examples of knowledge base communication with programmers

5.5.1 Example 1: Searching for information on Aglets

In the first example, it is assumed that the novice mobile agent programmer requires information on *Aglets*, which may either be a topic or a keyword. The proposed knowledge base environment is activated and the programmer navigates through the system as follows:



This knowledge base is intended to introduce novice mobile agent programmers to the mobile agent environment and to serve as a reference tool during the process of designing and developing a mobile agent system. It can be used in conjunction to the accompanying dissertation or as a stand-alone implementation.

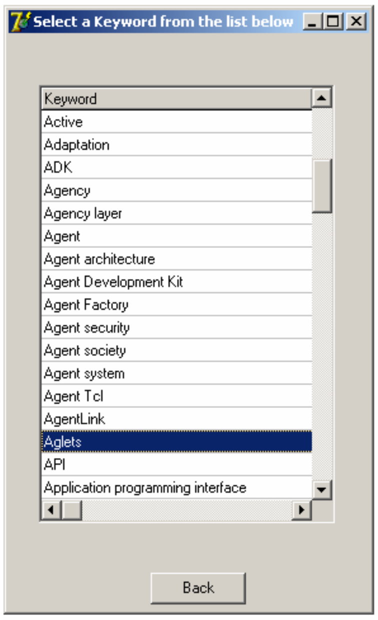
When the user choose either a topic or a keyword, the knowledge base displays the following:

- The relevant section in the dissertation
- A short explanation of the concept according to the topic or keyword
- An example, if available
- References to related articles or websites

Search through the database by selecting Topic or Keyword below

Figure 5.10 Select the *Keyword* button

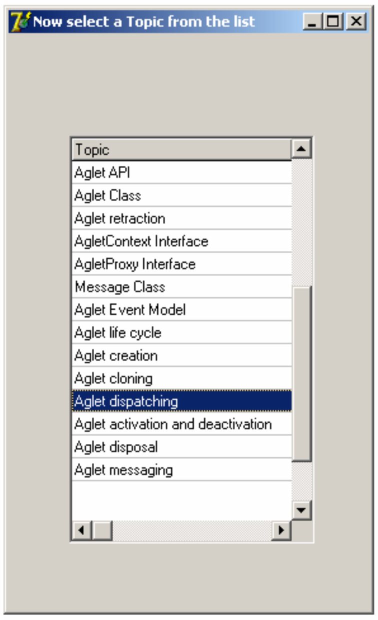
The *Keyword* button is picked on the initial user interface shown in Figure 5.10 to initiate a search for keywords. This causes a scrolling menu of all the available keywords in the knowledge base to be displayed, as shown in Figure 5.11.



Keyword
Active
Adaptation
ADK
Agency
Agency layer
Agent
Agent architecture
Agent Development Kit
Agent Factory
Agent security
Agent society
Agent system
Agent Tcl
AgentLink
Aglets
API
Application programming interface

Figure 5.11 Select *Aglets* from the list of keywords

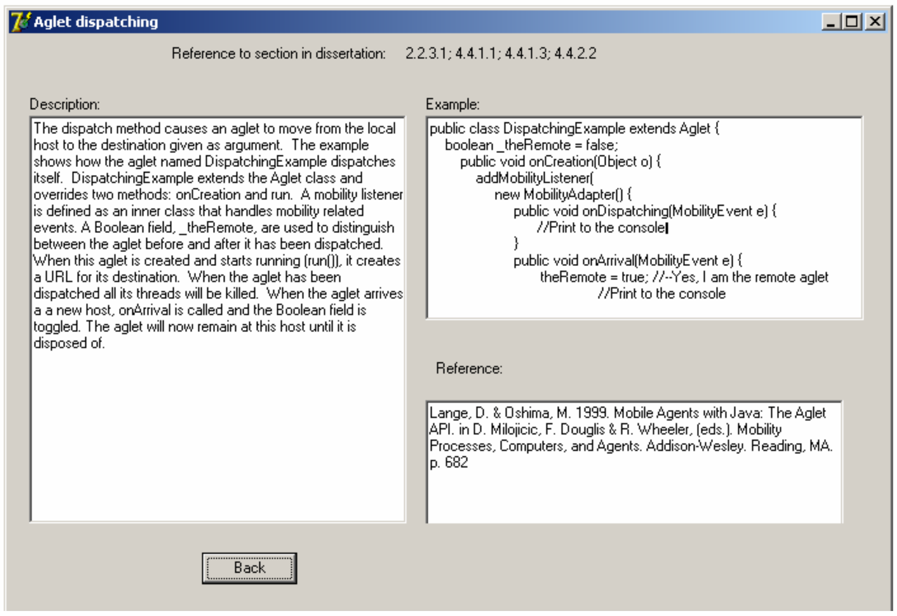
The user may use either the scroll bar or the down arrow on the keyboard to scroll through the list of available keywords. Should he/she not find an appropriate keyword, he/she may return to the initial user interface by picking the *Back* button. In this case the scroll bar or down arrow is used to move the cursor to the keyword *Aglets* (see Figure 5.11). The user selects the keyword by pointing with the mouse to initiate a search for topics linked to the keyword *Aglets*. This causes an abbreviated list of topics, linked to the keyword *Aglets*, to be retrieved from the knowledge base and displayed, as shown in Figure 5.12.



Topic
Aglet API
Aglet Class
Aglet retraction
AgletContext Interface
AgletProxy Interface
Message Class
Aglet Event Model
Aglet life cycle
Aglet creation
Aglet cloning
Aglet dispatching
Aglet activation and deactivation
Aglet disposal
Aglet messaging

Figure 5.12 List of topics with *Aglet* as a keyword

As shown in Figure 5.12, the topic *Aglet dispatching* is picked, causing the relevant information in the knowledge base about the dispatching of Aglets to be displayed (see Figure 5.13).



Reference to section in dissertation: 2.2.3.1; 4.4.1.1; 4.4.1.3; 4.4.2.2

Description:

The dispatch method causes an aglet to move from the local host to the destination given as argument. The example shows how the aglet named DispatchingExample dispatches itself. DispatchingExample extends the Aglet class and overrides two methods: onCreation and run. A mobility listener is defined as an inner class that handles mobility related events. A Boolean field, _theRemote, are used to distinguish between the aglet before and after it has been dispatched. When this aglet is created and starts running (run()), it creates a URL for its destination. When the aglet has been dispatched all its threads will be killed. When the aglet arrives a a new host, onArrival is called and the Boolean field is toggled. The aglet will now remain at this host until it is disposed of.

Example:

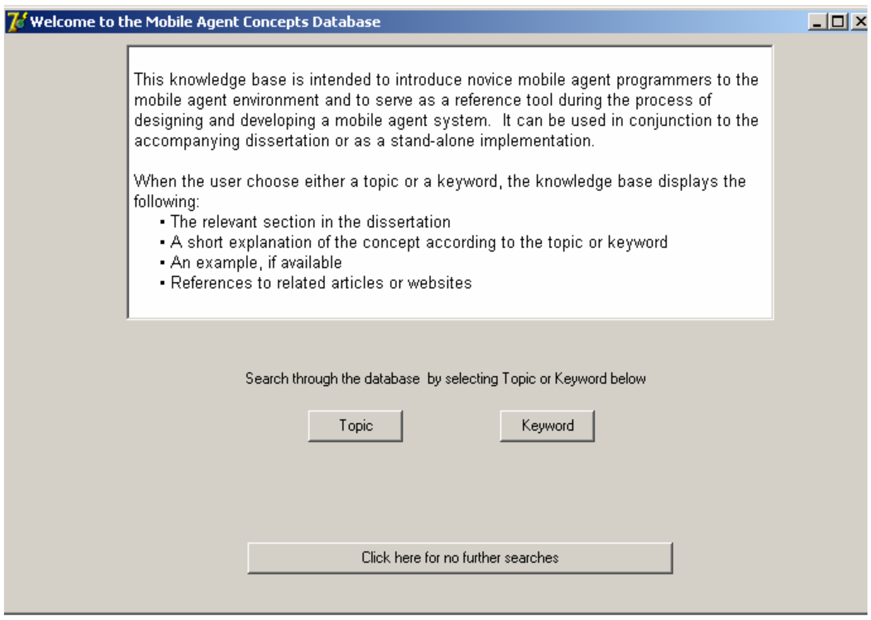
```
public class DispatchingExample extends Aglet {
   boolean _theRemote = false;
      public void onCreation(Object o) {
         addMobilityListener(
            new MobilityAdapter() {
               public void onDispatching(MobilityEvent e) {
                  //Print to the console
               }
               public void onArrival(MobilityEvent e) {
                  theRemote = true; //--Yes, I am the remote aglet
                         //Print to the console
```

Reference:

Lange, D. & Oshima, M. 1999. Mobile Agents with Java: The Aglet API. in D. Milojicic, F. Douglis & R. Wheeler, (eds.). Mobility Processes, Computers, and Agents. Addison-Wesley. Reading, MA. p. 682

Figure 5.13 Information about the dispatch method for Aglets is displayed under the topic *Aglet dispatching*

The *Reference to section in dissertation* note indicates all the sections in the dissertation that refer to Aglets, though not all of them mention the dispatch method. Clarification about the `dispatch` method is displayed in the *Description* text box while the code for the method itself is displayed in the *Example* text box. Note that the user may click inside the *Example* text box and use the down arrow to scroll in order to view the complete example. The reference in the *Reference* text box refers the programmer to more information about the `dispatch` method. Picking the *Back* button causes the initial user interface to be displayed, as shown in Figure 5.14.


Welcome to the Mobile Agent Concepts Database

This knowledge base is intended to introduce novice mobile agent programmers to the mobile agent environment and to serve as a reference tool during the process of designing and developing a mobile agent system. It can be used in conjunction to the accompanying dissertation or as a stand-alone implementation.

When the user choose either a topic or a keyword, the knowledge base displays the following:

- The relevant section in the dissertation
- A short explanation of the concept according to the topic or keyword
- An example, if available
- References to related articles or websites

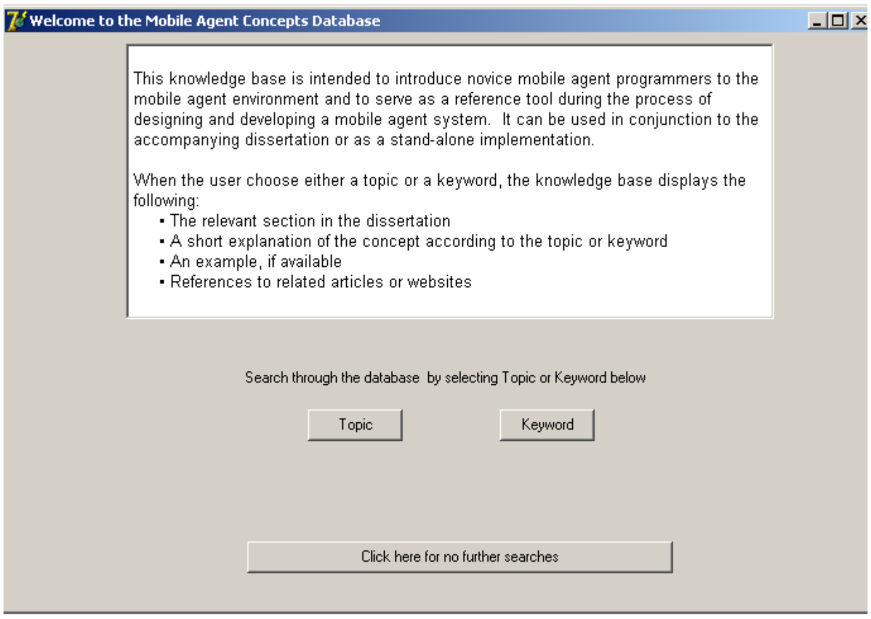
Search through the database by selecting Topic or Keyword below

Figure 5.14 Initial user interface

From the initial user interface, the programmer may continue with further searches based on a specific topic or keyword. Alternatively, the button *Click here for no further searches* may be selected to exit the system by pointing with the mouse.

5.5.2 Example 2: Searching for information on BDI agents

In the second example, one assumes that the novice mobile agent programmer requires information on BDI agents, which may either be a topic or keyword. In this instance, one assumes the programmer will do a search on the list of available topics. Again, the proposed knowledge base environment is activated as shown in Figure 5.15 and the programmer navigates through the system as follows: The *Topic* button is chosen from the initial user interface by pointing with the mouse. This causes a scrolling menu of all the available topics in the knowledge base to be displayed, as shown in Figure 5.16.



Welcome to the Mobile Agent Concepts Database

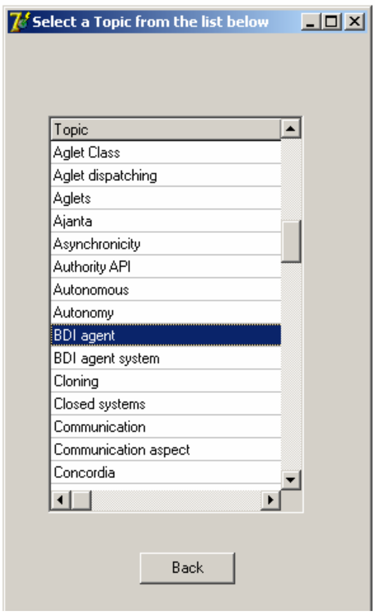
This knowledge base is intended to introduce novice mobile agent programmers to the mobile agent environment and to serve as a reference tool during the process of designing and developing a mobile agent system. It can be used in conjunction to the accompanying dissertation or as a stand-alone implementation.

When the user choose either a topic or a keyword, the knowledge base displays the following:

- The relevant section in the dissertation
- A short explanation of the concept according to the topic or keyword
- An example, if available
- References to related articles or websites

Search through the database by selecting Topic or Keyword below

Figure 5.15 Choose the *Topic* button to activate the display of a list of available topics



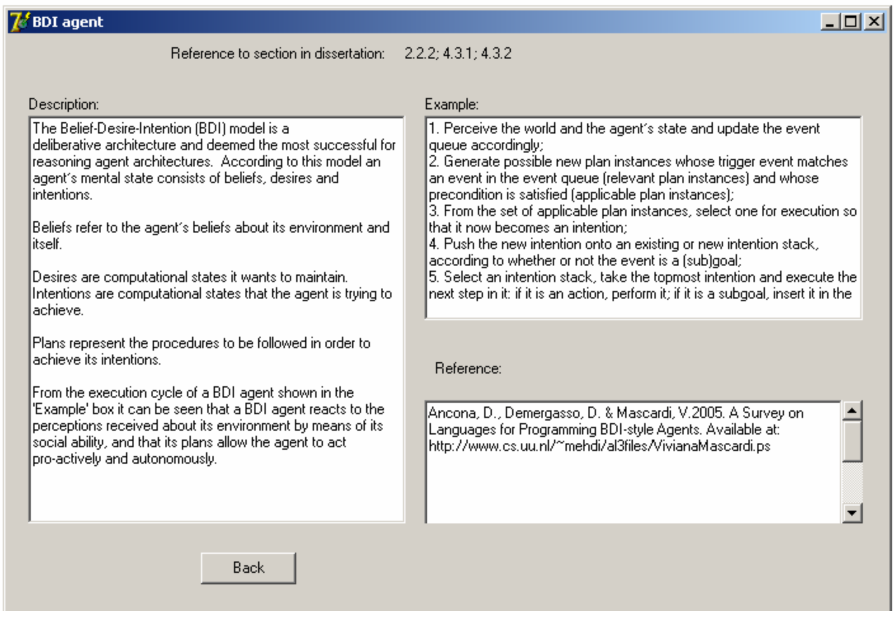
Select a Topic from the list below

Topic
Aglet Class
Aglet dispatching
Aglets
Ajanta
Asynchronicity
Authority API
Autonomous
Autonomy
BDI agent
BDI agent system
Cloning
Closed systems
Communication
Communication aspect
Concordia

Figure 5.16 Pick *BDI agent* from the list of available topics

The programmer uses the scroll bar or the down arrow on the keyboard to scroll through the list of topics. Should he/she not find an appropriate topic, he/she may return to the initial user interface by selecting the *Back* button. In this case, the down arrow is used to move the cursor to the topic *BDI* *agent.* Selecting this topic by pointing with the mouse as shown in Figure 5.16, activates the display of the information on the searched topic in the knowledge base. The result is shown in Figure 5.17.

The *Reference to section in dissertation* note indicates all the sections in the dissertation that refer to the BDI model. Clarification about the BDI model itself is displayed in the *Description* text box while the execution cycle of a BDI agent is displayed in the *Example* text box. Note that the user may click inside the *Example* text box and use the down arrow to scroll in order to view the complete execution cycle. The references in the *Reference* text box indicate articles where more information on BDI agents can be found. The scroll bar to the right of the *Reference* text box shows that more than one reference is available for the topic *BDI agent*. Again, selecting the *Back* button by pointing with the mouse causes the initial user interface to be displayed (not shown here), from where the *Click here for no further searches* button may be selected to exit the system.



Reference to section in dissertation: 2.2.2; 4.3.1; 4.3.2

Description:

The Belief-Desire-Intention (BDI) model is a deliberative architecture and deemed the most successful for reasoning agent architectures. According to this model an agent´s mental state consists of beliefs, desires and intentions.

Beliefs refer to the agent´s beliefs about its environment and itself.

Desires are computational states it wants to maintain. Intentions are computational states that the agent is trying to achieve.

Plans represent the procedures to be followed in order to achieve its intentions.

From the execution cycle of a BDI agent shown in the 'Example' box it can be seen that a BDI agent reacts to the perceptions received about its environment by means of its social ability, and that its plans allow the agent to act pro-actively and autonomously.

Example:

1. Perceive the world and the agent´s state and update the event queue accordingly;
2. Generate possible new plan instances whose trigger event matches an event in the event queue (relevant plan instances) and whose precondition is satisfied (applicable plan instances);
3. From the set of applicable plan instances, select one for execution so that it now becomes an intention;
4. Push the new intention onto an existing or new intention stack, according to whether or not the event is a (sub)goal;
5. Select an intention stack, take the topmost intention and execute the next step in it: if it is an action, perform it; if it is a subgoal, insert it in the

Reference:

Ancona, D., Demergasso, D. & Mascardi, V.2005. A Survey on Languages for Programming BDI-style Agents. Available at: http://www.cs.uu.nl/~mehdi/al3files/VivianaMascardi.ps

Figure 5.17 Display of information on topic *BDI agent*

5.6 Testing the Knowledge Base

A small number (fewer than 10) of novice agent programmers were invited to experiment with the knowledge base. They were given a simple assignment to answer the question “What comprises the life cycle of a mobile agent?” The test subjects were observed while they performed the task, and notes were made on their performance and behaviour. During the subsequent interview with each subject, the subjects rated the usability of the knowledge base according to the ISO standards of effectiveness, efficiency and satisfaction (*Usability*, 2005) on a scale of 1 to 10. They also gave feedback on their experience of using the knowledge base.

The questions used to explain the concepts *effectiveness*; *efficiency* and *satisfaction* to test subjects are shown in Table 5.1.

Concept	**Questions explaining the concept**
Effectiveness	"Does the knowledge base do what it says it does?" "Did you find the information you were looking for?"
Efficiency	"How quickly did you manage to find the information?" "Would it have been quicker to do an Internet search?"
Satisfaction	"Did the knowledge base satisfy the purpose?" "Were you satisfied with the information you found?"

Table 5.1 Questions used to explain usability concepts

Some of the test subjects preferred using the *Topic* button on the initial user interface to initiate searches, while others preferred the *Keyword* button for this purpose. Nevertheless, the test subjects rated both the effectiveness and efficiency of the knowledge base at an average of 8 out of 10. They expressed a level of satisfaction of 7.5 out of 10. All the test subjects agreed that they would use it as a quick referencing tool, and would recommend it as an introduction to someone interested in mobile agents. Most of the test subjects concurred that the knowledge base would assist them in mastering concepts related to mobile agents and in programming a mobile agent. The references in particular were regarded as valuable. The layout was seen as easy to use and acceptable, though incorporating more colour, as well as consulting HCI guidelines for interfaces was suggested.

Additional features expected from such a tool include facilities to print, extract and store information. Typing in the first letters of a topic or keyword instead of browsing down is preferred. Similarly test subjects wish to move up one level instead of two levels after a topic selected from the abbreviated topic list has been displayed. Links to keywords or key concepts from the description text box would enhance the usability of the knowledge base. All test subjects agree that, should the knowledge base be exclusively for their own use, they would use a facility that supports their personal knowledge management. Such a facility could allow them to make notes, to add to the contents of the knowledge base or to edit it, obviously with certain privileges, in order to enhance their own insight and comprehension. An interesting suggestion is to add an intelligent environment providing templates for programming mobile agents. More real-life examples and more code examples will also enhance the contents of the knowledge base. All of these features could easily be added.

5.7 Conclusion

This chapter discusses the development and usage of the knowledge base environment that was developed to assist novice mobile agent programmers in order to allow them to develop commercially acceptable mobile agent systems. The value of the knowledge base lies in reducing the time and effort to master the considerable number of concepts in the mobile agent environment, while also serving as a referencing aid. Though the knowledge base could only be tested to a limited extent, the outcome of the test confirms the value of the knowledge base, since all of the test subjects involved in testing it regard it as an eminently usable tool. This consent is expressed both verbally and in the usability ratings they ascribe to the knowledge base.

The comments and suggestions stated by the test subjects offer directions for future improvements to the knowledge base and the knowledge base environment. The comments and suggestions refer largely to the user-friendliness and ease of use of the knowledge base. It is clear that both the *contents* to be presented in the knowledge base and the *manner* in which it is presented are crucial to the usability of the knowledge base. Furthermore, when deploying the knowledge base, ideally each user should receive a copy for his/her exclusive use, with a facility to develop the contents of the knowledge base further.

In conclusion, while this knowledge base has limitations, which are addressed in the following chapter, the test subjects nevertheless certainly regard it as a valuable tool to assist novice mobile agent programmers in developing commercially acceptable mobile agent systems. The concept of the knowledge base thus achieves its purpose of transferring knowledge to novice mobile agent programmers.

In Chapter 6 the research results, their significance and limitations are discussed. The research is also compared to similar research and future research areas are identified.

CHAPTER 6

6 CONCLUSION

6.1 Introduction

This chapter provides the conclusion to the research. A summary of the objectives with the research and the realization thereof is provided in section 6.2. Section 6.3 discusses the research results, and section 6.4 compares it with similar research. The significance of the research is pointed out in section 6.5 and its limitations in section 6.6. Future research is discussed in section 6.7 and the final conclusion is made in section 6.8.

6.2 Summary of objectives and realization thereof

The benefits that mobile agents can offer to current trends in computing, such as pervasiveness, mobile computing, the Semantic Web and Web Services, are hampered by problems pointed out in section 1.1, owing to the independent development of agent and mobile agent systems. These problems (lack of an agreed definition, duplication of effort, inability to satisfy industrial strength requirements and incompatibility) are aggravated by the fact that more than one community are active in the mobile agent field, with differing views on mobile agents and their abilities. Nevertheless, considering the rapid expansion of Internet applications, trends such as mobile computing, and the benefits offered by mobile agents for these trends, it seems that introducing novices to programming mobile agents may become a regular task in the near future. As pointed out earlier, this indicates a need for a comprehensive programming model to serve as an introduction to novice mobile agent programmers in mastering the paradigm/milieu.

The outcome of this research is to develop a knowledge base that aids novice mobile agent programmers in making the required paradigm shift by providing the conceptual background as well as a framework for constructing mobile agent systems. To this end, the following set of goals to be achieved by the knowledge base is identified. It should

1. explain what mobile agents are;
2. elaborate on their characteristics and the environment in which they operate;
3. indicate the circumstances under which using mobile agents will be appropriate;
4. discuss the context for programming mobile agents;
5. recommend a framework for constructing a mobile agent system; and

6. supply a set of guidelines for programming mobile agents with the focus on programming concepts.

Chapter 2 achieves the first three goals mentioned. This chapter defines a mobile agent as an active entity that can migrate autonomously through a computer network and resume execution at a remote host. The historic development of mobile agents, the communities active in the field and the different views held by these communities introduce the mobile agent milieu. The main criteria for the use of mobile agent applications, namely limited network capacity and circumstances requiring asynchronous execution, autonomy and persistence, are based on their salient characteristics. The mobile agent execution environment is introduced, and available mobile agent standards are described.

In order to provide insight into the context and the new paradigm required for programming mobile agents, Chapter 3 describes both the agent orientation paradigm and the OO paradigm. Agent orientation is shown to be an extension of OO by pointing out the difference between agents and objects. Agents can be implemented using specific agent technologies such as agent-oriented programming, as well as more general technologies such as object technology. Guidelines on when to use agent orientation and the benefits that mobile agents hold for these types of systems are pointed out.

Chapter 4 addresses the fifth and sixth goals stated. A generic mobile agent system architectural model presents a comprehensive overview of the essential architectural components required to implement a mobile agent system, while incorporating agent orientation. Guidelines for programming mobile agents systems are included in the descriptions of the layers composing the architectural model.

Chapter 5 presents an example of a visualization of the knowledge base that was developed to contain the concepts introduced in Chapters 2 to 4. It discusses the design and implementation of the visualization, provides some examples of interaction with the knowledge base and highlights its importance. The interface supplies references to this dissertation as well as references to literature that expand on the concepts summarized in the knowledge base. It also serves as a quick reference tool to refresh concepts while novices to the mobile agent paradigm explore the literature and implement mobile agent systems.

6.3 Discussion of research results

Programming agents require the application of different programming principles from those used in current programming paradigms such as OO. As pointed out in Chapters 3 and 4, a paradigm shift is required in order to develop commercially acceptable mobile agent systems. However, it can take a significant amount of time and programming effort before novice mobile agent programmers realize this. The essential knowledge required to enable novices to make this paradigm shift is thus addressed in Chapters 2 to 4, but mainly in Chapter 3. This can be used as a source for example for a fourth-year course in mobile agent programming. In so doing, novices' learning curve would improve noticeably, while unproductive programming effort will be reduced at the same time. The experience of students engaged in Project TUTA provides an example of the value such a knowledge base could offer.

The comprehensive overview provided by the generic mobile agent system architectural model takes agent orientation into account. When the novice uses this as a base for the design and implementation of a mobile agent system, he/she will include all the essential architectural elements required in his/her system.

Developing a knowledge base is widely recognized as an instrument to share knowledge. Furthermore, there is a strong technical and economic imperative to share information through the use of knowledge bases, thereby reducing one of the most costly aspects of the company budget, namely salaries. Test subjects who experimented with the visual example of the proposed knowledge base all agree that they would use it to assist them in mastering concepts related to mobile agents and in programming mobile agents. As such it offers a valuable quick referencing tool.

From the above, it becomes clear that the research questions of this dissertation have indeed been answered. The question that was asked in the first chapter was *"Since novice mobile agent programmers[13] require a paradigm shift to construct successful systems, how can they be equipped to grasp the contextual issues and gain the necessary skills within reasonable time limits?"*

[13]Novices include mature, beginner and intermediate programmers who have no experience of mobile agent programming.

This research provides three structures whereby the research question is answered. The first structure is a complete reference to equip the novice mobile agent programmer with contextual information and knowledge of mobile agent systems development. At the same time the novice is introduced to agent orientation, which enables him/her to make the paradigm shift needed to apply the proper principles involved in agent programming in order to realize mobile agents' full potential. The reference represents a summary of a few years' worth of literature study. This structure can be valuable in teaching mobile agent programming to graduate students or in a graduate computing project.

The second structure is the presentation of a generic mobile agent system architecture that can be used by novice programmers to design a mobile agent system while taking agent orientation into account. It follows a layered approach that isolates the individual components that each forms an essential part of the mobile agent system. In this way, while providing a broad overview of the important system elements during design, the approach also enables developers to focus on the tasks pertaining to a specific layer, which forms a functional unit. At the same time, the layered model facilitates the development of layers in parallel, while implementing each layer independently also assists program maintenance, debugging and upgrading.

The third structure provides a visualization of a knowledge base that unlocks concepts and knowledge units to the novice mobile agent programmer while programming. In a multi-window operating system, the programmer can merely activate the knowledge base interface and keep it active in the background while working in a programming language environment.

6.4 Comparison with other similar research

In comparison to other research, this research is focused on *concepts* important to the development of mobile agent systems. Other research is frequently more practically oriented, resulting in programming code. The result of this research is therefore more fundamental as it aims to point out essential principles.

Examples of more practically oriented research groups are those involved with projects such as D'Agents (*D'Agents: Mobile Agents at Dartmouth College*, 2002), Agent Factory (*Agent Factory*, 2003), Tryllian's ADK (*Agent Development Kit*, 2004) and Voyager ® (*Recursion Software - Intelligent Mobile Agent Technology*, 2004). They typically offer papers, tutorials, links to other mobile agent systems, conference proceedings and other publications, training courses, lists of frequently asked questions (FAQs) or even software engineering consultants to assist novices and developers.

As was pointed out in Chapter 4, the generic mobile agent system architectural model is more comprehensive than the models used to identify components to be included. It is also more comprehensive than other known models, such as those resented by Braun *et al.* (2001), Johansen et al. (2002) and Kendall *et al.* (2000).

No tool similar to the knowledge base could be found during an investigation into the manner in which such research groups and educational institutions introduce novices to the mobile agent paradigm. Lists of FAQs offer perhaps the nearest concept to it. However, FAQs do not offer a comprehensive overview and are often focused on programming code rather than on fundamental concepts. Furthermore, depending on the list of FAQs, it may not be as easy to search for a specific concept in the FAQs as using the knowledge base is. In conjunction with the dissertation and the generic mobile agent system architectural model, the knowledge base could provide a substantially quicker and broader overview than the methods currently employed.

6.5 Significance

The significance of this research lies in pointing out the paradigm shift required to enable novices to become mobile agent programmers, and providing an architectural model to assist the novice during mobile agent system development in keeping with this paradigm shift.

The significance of certain elements, such as the generic mobile agent system architectural model (section 4.4) and the knowledge base for novice mobile agent programmers (section 5.2), has already been discussed in detail. Only a brief summary of the significance of the architectural model and the knowledge base are therefore reiterated.

The architectural components represent the essential aspects of a mobile agent system in a simplified, general but clear way, while relating it to agent orientation. In so doing the novice mobile agent programmer is presented with a basis for making sensible decisions when designing and implementing mobile agents, as well as for choosing a mobile agent toolkit. Since the architectural model is generic, it can be transferred to any specific application.

The comprehensive survey in Chapters 2 to 4 and the broad overview presented by the essential requirements for mobile agent systems in the architectural model allow the novice mobile agent programmer to *orientate* himself or herself to the mobile agent field at both the mobile agent and the mobile agent system level. The knowledge base assists the novice mobile agent programmer both in grasping and refreshing the key concepts in the mobile agent paradigm and in placing it in context during the process of acquainting himself/herself with the field. The knowledge base can also be used as a tool to recap concepts and to provide references while designing and developing a mobile agent system.

Including agent orientation in the portrayal of the mobile agent execution environment offers a strong theoretical base enabling novices to develop mobile agent systems according to the appropriate principles. The guidelines incorporated in the generic mobile agent system architecture further aid the development of commercially viable systems.

6.6 Limitations

The generic mobile agent system architectural model should be practically implemented and tested to prove its worth. Since the purpose of the architectural model is to provide a comprehensive view of the essential elements required in a mobile agent system, it is of a theoretical nature. Its focus is thus on conveying relevant issues to be considered in each of the architectural components in order to enhance the design and development of mobile agent systems.

6.7 Future research

A number of areas for future research can be identified. Such efforts include research relating to the knowledge base, the architectural model, mobile agent toolkits and agent orientation.

The effectiveness of the paradigm shift and the accompanying architectural model in assisting novices should be determined. Such research has to be done in combination with researchers in education and educational psychologists in order to develop proper measuring instruments to determine the efficiency of novice mobile agent programmers after using the architectural model in conjunction with making a paradigm shift to agent orientation. Should the current visualization of the knowledge base be used in such research, HCI experts will be included in the effort to ensure that the interface provides optimal user-friendliness. Testing the effectiveness of the paradigm shift and the accompanying architectural model would take at least a year, since an extended testing period is required. During this period one group of students would typically develop a mobile agent system by using the architectural model while making the required paradigm shift, while the control group would develop the same system without having access to the architectural model or information on the paradigm shift. Taking the development of the measuring instruments, designing of the tests, and the actual testing of the students into account, shows that this research would require at least eighteen months.

In order to prove the worth of the generic mobile agent system architectural model, it should be practically implemented and tested to complete it.

With regard to mobile agent toolkits, future research could investigate various mobile agent toolkits to determine which would be suitable to introduce novices to programming mobile agents. Determining which agent and mobile agent principles are included in the various toolkits, and how clearly these principles can be identified in the toolkit, could be one of the criteria for such an objective.

Research areas identified in agent orientation include investigating agent programming languages and constructs in agent programming languages specifically aimed at mobile agents, investigating the gap between AOSE and the implementation of agents/mobile agents, and investigating extensions to agent-oriented methodologies to include the analysis and design of mobile agents. Investigating the relationship between software components and agents, as well as implementing mobile agents by with software components would also be a worthwhile pursuit.

6.8 Conclusion

In the light of current trends in computing, and mobile agents' applicability to these trends, a real need for tools to assist novices in mastering the programming of mobile agents has been identified and addressed in this research. The introduction to the required paradigm shift combined with the architectural model and the accompanying knowledge base present an example of one such tool.

Bibliography

2004 IEEE International Conference on Mobile Data Management (MDM'04), Available at: http://www.cs.duke.edu/mdm2004/. [Accessed 25/5/2005].

About Distributed Agents (2004), Available at: http://dsonline.computer.org/agents/about.htm. [Accessed 7/12/2004].

Agent Development Kit (2004), Available at: http://www.tryllian.com/technology/product1.html#. [Accessed 8/2/2005].

Agent Factory (2003), Available at: http://www.agentfactory.com/agentfactory/index.htm. [Accessed 8/2/2005].

AgentBuilder (2004), Available at: http://www.agentbuilder.com/Documentation/whyAgents.html. [Accessed 2/11/2004].

AgentLink III, Available at: http://www.agentlink.org/. [Accessed 20/3/2004].

Ambler, S. W. (2001), *The Object Primer,* 2nd edn, Cambridge University Press, New York.

Amor, M., Fuentes, L. & Vallecillo, A. (2004), Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA*.

Ancona, D., Demergasso, D. & Mascardi, V. (2005), *A Survey on Languages for Programming BDI-style Agents*, Available at: http://www.cs.uu.nl/~mehdi/al3files/VivianaMascardi.ps. [Accessed 10/5/2005].

Aridor, Y. & Lange, D. B. (1998), Agent Design Patterns: Elements of Agent Application Design, *Proceedings of the Second International Conference on Autonomous Agents*: 108-115.

Aridor, Y. & Oshima, M. (1998), Infrastructure for Mobile Agents: Requirements and Design, *MA '98, LNCS 1477.*: 38-49.

Ashri, R. & Luck, M. (2001), *Towards a layered approach for agent infrastructure: the right tools for the right job*, Available at: http://www.ecs.soton.ac.uk/~mml/papers/aa01wsinf.pdf. [Accessed 12/8/2005].

Ashri, R. & Luck, M. (2002), Infrastructure Support for Agent-based Development, in *Foundations and Applications of Multi-Agent Systems. Lecture Notes in Artificial Intelligence*, vol. 2403, Springer-Verlag, 73-88.

Assertions and Protocol for the OASIS Security Assertion Markup Language 3 (SAML) V1.1.4 (2003), Available at: http://www.oasis-open.org/committees/download.php/1894/sstc-saml-core-1.1-draft-10.pdf. [Accessed 24/10/2005].

Badjonski, M., Ivanovic, M. & Budimac, Z. (2005), Adaptable Java Agents (AJA) - A Tool for Programming of Multi-Agent Systems, *ACM SIGPLAN Notices,* 40(2): 17-26.

Bauer, B. & Muller, J. (2004), Methodologies and Modeling Languages, in M. Luck, R. Ashri & M. D'Inverno, (eds.), *Agent-Based Software Development*, Artech House, Norwood MA, 77-132.

Bauer, B., Muller, J. P. & Odell, J. (2001), Agent UML: A Formalism for Specifying Multiagent Interaction, in P. Ciancarini & M. Wooldridge, (eds.), *Agent-oriented Software Engineering*, Springer-Verlag, Berlin, 91-103.

Baumann, J. (2000), *Mobile Agents: Control Algorithms*, Springer-Verlag, Berlin Heidelberg New York.

Bellifemine, F., Caire, G., Poggi, A. & Rimassa, G. (2003), JADE A White Paper, *exp,* 3(3): 6-19.

Bierman, E. & Cloete, E. (2002), Classification of Malicious Host Threats in Mobile Agent Computing, in *South-African Institute of Computer Science and Information Technology 2002*, Port Elizabeth, South-Africa, 141-148.

Booch, G. (1994), *Oject-Oriented Analysis and Design with Applications*, Addison-Wesley.

Brantschen, S. & Haas, T. (2002), *Agents in a J2EE World*, Available at: http://www.whitestein.com/resources/whitepapers/whitestein_agentsj2ee.pdf. [Accessed 12/8/2005].

Braubach, L. & Pokahr, A. (2002), *Jadex BDI Agent System*, Available at: http://vsis-www.informatik.uni-hamburg.de/projects/jadex/features.php. [Accessed 10/3/2005].

Braubach, L. & Pokahr, A. (2004), *Goal Oriented Programming*, Available at: http://www.cs.uu.nl/~mehdi/al3tf8/Pokahr.pdf. [Accessed 3/8/2004].

Braubach, L., Pokahr, A. & Lamersdorf, W. (2004a), Jadex: A Short Overview, *Main Conference Net.ObjectDays 2004, AgentExpo*: 13.

Braubach, L., Pokahr, A., Lamersdorf, W. & Moldt, D. (2004b), Goal Representation for BDI Agent Systems, in *The Second International Workshop on Programming Multiagent Systems (PROMAS-2004)*, eds. R. H. Bordini, M. Dastani, J. Dix & A. E. Fallah-Seghrouchni, New York, NY, USA, 9-20.

Braun, P., Eismann, J., Erfurth, C. & Rossak, W. (2001), TRACY A Prototype of an Architected Middleware to support Mobile Agents, in *Eight Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'01)*, IEEE, 6.

Burmeister, B. (1996), Models and Methodology for Agent-Oriented Analysis and Design, in K.Fischer, (ed.) *Working Notes of the KI '96 Workshop on Agent-Orientated Programming and Distributed Systems*, DFKI.

Cabri, G., Ferrari, L. & Leonardi, L. (2005), Injecting roles in Java agents through runtime bytecode manipulation, *IBM Systems Journal,* 44(1): 185-208.

Cabri, G., Leonardi, L. & Zambonelli, F. (2000), Agents for information retrieval: Issues of mobility and coordination, *Journal of Systems Architecture,* 46(15): 1419-1433.

Calisti, M. (2003), FIPA standards for promoting interoperability of industrial agent systems, in *FIPA Presentation, Agentcities, Information Days*, Barcelona, Spain.

Center for Mobile Computing (2005), Available at: http://cmc.cs.dartmouth.edu/. [Accessed 11/5/2005].

Cervenka, R., Trencansky, I., Calisti, M. & Greenwood, D. (2005), AML: Agent Modeling Language Toward Industry-Grade Agent-Based Modeling, in J. Odell, P. Giorgini & J. P. Muller, (eds.), *Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004,*, vol. LNCS 3382/2005, Springer-Verlag Berlin Heidelberg, New York, NY, USA, 31-46.

Cloete, E. & Gerber, A. (2004), OO Systems Development Barriers for Structural Developers, in *Proceedings of the 6th International Conference on Enterprise Information Systems*, vol. III, 42-47.

Collier, R. W., Rooney, C. F. B., O'Donoghue, R. P. S. & O'Hare, G. M. P. (2000), Mobile BDI Agents, in *Proceedings of the 11th Irish Conference on Artificial Intelligence & Cognitive Science*, Galway, Ireland.

Comet Way Java Agent Kernel (2005), Available at: http://www.agentkernel.com/. [Accessed 9/5/2005].

Corradi, A. (2001), Mobile agent technology: from first proposals to current evolutions, *Microprocessors and Microsystems,* 2(2): 63-64.

Cossentino, M., Burrafato, P., Lombardo, S. & Sabatucci, L. (2002), *Introducing Pattern Reuse in the Design of Multi-Agent Systems*, Available at: http://citeseer.ist.psu.edu/cache/papers/cs/26501/http:zSzzSzwww.csai.unipa.itzSzcossentinozSzpaperssZsAITA02.pdf/cossentino021introducing.pdf. [Accessed 3/8/2004].

Cuadron, M., Groote, J. F., Van Hee, K., Hemerik, K., Somers, L. & Verhoeff, T. (2004), *Software Engineering Reference Framework*, Available at: http://www.win.tue.nl/~jfg/articles/CSR-04-39.pdf. [Accessed 26/5/2005].

Da Silva, A. R., Da Silva, M. M. & Romão, A. (2000), *Web-based Agent Applications: User Interfaces and Mobile Agents*, Available at: http://citeseer.nj.nec.com/499980.htm. [Accessed 24/10/2005].

D'Agents: Mobile Agents at Dartmouth College (2002), Available at: http://agent.cs.dartmouth.edu/. [Accessed 11/5/2005].

Dale, J. (1997), *A Mobile Agent Architecture for Distributed Information Management*, Ph D, University of Southampton, Southampton.

Dastani, M. (2004), *AgentLink-III Technical Forum Group Programming MultiAgent System PROMAS*, Available at: http://www.cs.uu.nl/~mehdi/al3tf8/report.pdf. [Accessed 11/4/2005].

Dastani, M. & Gomez-Sanz, J. J. (2005), *Programming Multi-Agent Systems A Report of the technical forum meeting*, Available at: http://www.cs.uu.nl/%7Emehdi/tfg/al3files/report.pdf. [Accessed 3/8/2005].

Delamaro, M. & Picco, G. P. (2002), Mobile Code in .NET: A Porting Experience, in N. Suri, (ed.) *Mobile Agents 2002, LNCS 2535*, Springer-Verlag, Berlin Hheidelberg, 2002.

A Dictionary of Computing (2004), Available at: http://0-www.oxfordreference.com.oasis.unisa.ac.za/views/GLOBAL.html. [Accessed 15/4/2005].

Ellamari, M. & Lalonde, W. (1999), An Agent-Oriented Mehodology: High-Level and Intermediate Models, in *1st International Workshop on Agent-Oriented Information Systems (AOIS 1999)*, Heidelberg, Germany, 17.

Errol, K., Lang, J. & Levy, R. (2000), Designing Agents from Reusable Components, in *Fourth International Conference on Autonomous Agents*, 76-77.

Fallah-Seghrouchni, A. E. & Suna, A. (2003a), CLAIM: A Computational Language for Autonomous, Intelligent and Mobile Agents, in *Programming multi-agent systems : first international workshop, ProMAS 2003*, eds. M. Dastani, J. Dix & A. E. F. Seghrouchni, Berlin; New York: Springer, Melbourne, Australia, 90-110.

Fallah-Seghrouchni, A. E. & Suna, A. (2003b), An Unified Framework for Programming Autonomous, Intelligent and Mobile Agents, in *Multi-Agent Systems and Applications III: 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003)*, eds. V. Marik, J. Muller & M. Pchouek, Springer-Verlag Berlin Heidelberg, Prague, Czech Republic, 353-362.

Farmer, W. M., Guttman, J. D. & Swarup, V. (1996), Security for Mobile Agents: Issues and Requirements, in *National Information systems Security Conference (NISSC)*.

Feridun, M. & Krause, J. (2001), A framework for distributed management with mobile components, *Computer networks,* 35: 25-38.

Finin, T., Labrou, Y. & Peng, Y. (1998), *Mobile Agents Can Benefit from Standards Efforts on Interagent Communication*, Available at: http://www.cs.umbc.edu/~finin//papers/IEEECommDraft.pdf. [Accessed 24/10/2005].

FIPA00087 - FIPA 98 Part 11 Version 1.0: Agent Management Support for Mobility Specification (1998), Available at: http://www.fipa.org/specs/fipa00087/. [Accessed 23/5/2005].

FIPA Methodology Technical Committee, Available at: http://www.fipa.org/activities/methodology.html. [Accessed 20/7/2005].

FIPA:The Foundation for Intelligent Agents (2000), Available at: http://www.fipa.org/specifications/index.html. [Accessed 12/10/2005].

Fisher, M. (1994), A Survey of Concurrent METATEM - The Language and Its Applications, in D. Gabbay & H. Ohlbach, (eds.), *Temporal Logic - Proceedings of the First International Conference (Lecture Notes in Artificial Intelligence 827)*, Springer, New York, 480-505.

Flores-Mendez, R. (1999), Towards a Standardization of Multi-Agent System Frameworks, *ACM Crossroads Student Magazine* (5.4): 1-10.

Fortino, G., Russo, W. & Zimeo, E. (2004), A statecharts-based software development process for mobile agents, *Information and Software Technology,* 46(13): 907-921.

Franklin, S. & Graesser, A. (1996), Is it an Agent, or just a Program? A taxonomy for Autonomous Agents, in *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag.

Gartner's Hype cycle, Available at: http://gsb.haifa.ac.il/~sheizaf/ecommerce/GartnerHypeCycle.html. [Accessed 1/12/2005].

Genesereth, M. R. & Finin, T. (1992), *Knowledge Interchange Format, Version 3.0 Reference Manual*, Available at: http://logic.stanford.edu/kif/Hypertext/kif-manual.html. [Accessed 24/10/2005].

Gerber, A. (2005), The Semantic Web, To be published, 26.

Gilbert, D., Aparacio, M., Atkinson, B., Brady, S., Ciccarino, J., Grosf, B., O'Connor, P., Osisek, D., Pritko, S., Spagna, R. & Wilson, L. (1995), *IBM Intelligent Agent Strategy*, IBM Corporation.

Giunchiglia, F., Mylopoulos, J. & Perini, A. (2002), The TROPOS Software Development Methodology: Processes, Models and Diagrams, in C. Castelfranchi & W. Johnson, (eds.), *Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems AAMAS'02*, ACM Press, New York, 35-36.

Glaser, N. (1996), *Contribution to Knowledge Modelling in a Multi-Agent framework (the CoMoMAS Approach)*, PhD, L' Universite Henri Poincare, Nancy I, France.

Gray, R. S. (1996), Agent Tcl: A flexible and secure mobile-agent system, in *1996 Tcl/Tk Workshop*, USENIX Association, New York, 9-23.

Gray, R. S. (2004), Mobile Agents: Overcoming Early Hype and a Bad Name, in *2004 IEEE International Conference on Mobile Data Management (MDM'04)*, Los Alamitos, Calif., : IEEE Computer Society, Berkeley, California, 302-303.

Gray, R. S., Cybenko, G., Kotz, D., Peterson, R. A. & Rus, D. (2002), D'Agents: Applications and performance of a mobile agent system, *SOFTWARE: PRACTICE AND EXPERIENCE,* 35(6): 543-573.

Green, S., Hurst, L., Nangle, B., Cunningham, P., Somers, F. & Evans, R. (1997), *Software Agents: A review*, Available at: http://www.lsi.upc.es/~bejar/aia/aia-web/green97software.pdf. [Accessed 12/3/2005].

Grid computing (2006), Avaiable at http://en.wikipedia.org/wiki/Grid_computing. [Accessed 29/3/2006]

Grimley, M. J. & Monroe, B. D. (1999), *Protecting the Integrity of Agents: An Exploration into Letting Agents Loose in an Unpredictable World*, Available at: http://www.acm.org/crossroads/xrds5-4/integrity.html. [Accessed 13/4/2004].

Gschwind, T., Feridun, M. & Pleisch, S. (1999), ADK - Building Mobile Agents for Network and Systems Management from Reuseable Components, *Proceedings of First International Symposium on Agent Systems and Applications*: 13-21.

Harrison, C. G., Chess, D. M. & Kershenbaum, A. (1995), *Mobile Agents: Are they a good idea?*, IBM Research Report. IBM T. J. Watson Research Center.

Hayes-Roth, R. & Amor, D. (2003), *Radical Simplicity transforming Computers Into Me-Centric Appliances,* 1st edn, Prentice-Hall, Inc, New Jersey.

Henderson-Sellers, B. & Gorton, I. (2002), *Agent-based Software Development Methodologies*, Available at: http://www.open.org.au/Conferences/oopsla2002/. [Accessed 20/6/2005].

Hermann, K. & Zapf, M. (2000), *AMETAS White Paper Series*, Available at: http://www.accsis.de/ametas/haupt2.html. [Accessed 8/02/2005].

Horstmann, C. & Budd, T. (2005), *Big C++,* 1st edn, John Wiley & Sons, Inc.

Horvat, H., Cvetkovic, D., Milutinovic , D., Kocovic, P. & Kovacevic , V. (2000), Mobile Agents and Java Mobile Agent Toolkits, in *Proceedings of the 33rd Hawaii International Conference on System Sciences*, IEEE Computing Society, Los Alamos, CA., Maui, Hawaii, USA, 10.

Huber, M. J. (2000), JAM: A BDI-theoretic Mobile Agent Architecture, *AGENTLINK NEWS*(5): 3-6.

Huhns, M. N. (2004), *Software Development with Objects, Agents, and Services*, Available at: http://www.open.org.au/Conferences/oopsla2004/PaperAO/Keynote-Huhns.pdf. [Accessed 20/6/2005].

Huhns, M. N. & Singh, M. P. (1998), Agents and Multi-Agent Systems: Themes, Approaches, and Challenges, in M. N. Huhns & M. P. Singh, (eds.), *Readings in Agents*, Morgan Kaufmann Publishers, San Francisco, Calif., 1-23.

Iglesias, C. A., Garijo, M. & González, J. C. (2000), A Survey of Agent-Oriented Methodologies, in J. P. Müller, M. P. Singh & A. S. Rao, (eds.), *Lecture Notes in Computer Science*, vol. 1555/2000, Springer-Verlag Heidelberg, 317 - 330.

Iglesias, C. A., Garijo, M., Gonzalez, J. C. & Velasco, J. R. (1997), Analysis and design of multiagent systems using MAS-CommonKADS.

International Journal on Agent-Oriented Software Engineering (IJAOSE) (2005), Available at: http://www.inderscience.com/browse/index.php?journalID=174. [Accessed 24/6/2005].

Internet Key Exchange Security Protocol, Available at: http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113t/113t_3/isakmp.htm. [Accessed 1/11/2005].

JACK, Available at: http://www.agent-software.com. [Accessed 3/8/2005].

Jansen, W. A. (2000), Countermeasures for Mobile Agent Security, in *Computer Communications. Special issue on advanced security techniques for network protection*, Elsevier Science.

Jazayeri, M. & Lugmayr, W. (2000), Gypsy: A Component-based Mobile Agent System, *The proceedings of the Eight Euromicro Workshop on Parallel and Distributed Processing (EURO-PDP 2000), Rhodos, Greece*: 9.

Jennings, N. R. (1999a), Agent-Based Computing: Promise and Perils, in *16th Joint Conference on Artificial Intelligence (IJCAI-99)*, 1429-1436.

Jennings, N. R. (1999b), *Agent-Oriented Software Engineering*, Available at: http://www.ecs.soton.ac.uk/%7Enrj/download-files/cairo.pdf. [Accessed 20/3/2003].

Jennings, N. R. (2001), An Agent-Based Approach for Building Complex Software Systems, *Communcations of the ACM,* 44(4): 35-39.

Jo, C.-H. (2001), A Seamless Approach to the Agent Development, in *Proceedings of the 2001 ACM Symposium on Applied Computing*, 641-647.

Johansen, D. (2004), Mobile Agents: Right Concept, Wrong Approach, in *2004 IEEE International Conference on Mobile Data Management (MDM'04)*, Los Alamitos, Calif., : IEEE Computer Society, Berkeley, California, 300-301.

Johansen, D., Lauvset, K. J. & Marzullo, K. (2002), An extensible software architecture for Mobile Components, in *Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (EECBS'02)*, 231- 237.

Johansen, D., Renesse, R. v. & Schneider, F. B. (1995), *An introduction to the TACOMA distributed system version1.0*, University of Tromso, Tromso.

Juan, T., Pearce, A. R. & Sterling, L. (2002), ROADMAP: Extending the Gaia Methodology for Complex Open Systems, in *Fiirst International Joint Conference on Autonomous Agents and Multi Agent Systems AAMAS '02*, eds. C. Castelfranchi & W. Johnson, ACM Press, New York, 3-10.

Jumping Beans, Available at: http://www.jumpingbeans.com/. [Accessed 23/5/2005].

Kendall, E. A., Krishna, P. V., Suresh, C. B. & Pathak, C. V. (2000), An Application Framework for Intelligent and Mobile Agents, *ACM Computing Surveys,* 32(1es).

Kinny, D., Georgeff, M. & Rao, A. (1996), A methodology and modelling technique for systems of BDI agents, in W. v. d. Velde & J. Perram, (eds.), *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World MAAMAW'96 (LNCS Volume 1038)*, Springer, New York, 56-71.

Knowledge Base (2005), Available at: http://en.wikipedia.org/Knowledge_base. [Accessed 3/10/2005].

Knowledge Management (2005), Available at: http://en.wikipedia.org/wiki/Knowledge_management. [Accessed 14/11/2005].

Kotz, D., Gray, R. & Rus, D. (2002), *Future Directions for Mobile Agent Research*, Available at: http://dsonline.computer.org/0208/f/kot.htm. [Accessed 15/10/2002].

Kotz, D. & Gray, R. S. (1999), *Mobile Agents and the Future of the Internet*, Available at: http://www.cs.dartmouth.edu/~dfk/papers/kotz:future2/. [Accessed 20/5/2002].

Lall, M. (2005), *Selection of mobile agent systems based on mobility, communication and security aspects*, M.Sc., Unisa, Pretoria.

Lange, D. B. (1997), *Java Aglet Application Programming Interface White Paper - Draft 2*, Available at: http://www.trl.ibm.co.jp/aglets. [Accessed 12/10/2005].

Lange, D. B. (1998), Mobile Objects and Mobile Agents: the future of distributed computing? , in E. Jul, (ed.) *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'98), LNCS*, vol. 1445, Springer-Verlag, Brussels, Belgium, 1-12.

Lange, D. B. & Oshima, M. (1998), *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, Reading, M.A.

Lange, D. B. & Oshima, M. (1999), Seven Good Reasons for Mobile Agents, *Communications of the ACM,* 42(3): 88-89.

Lauvset, K. J. & Johansen, D. (2002), Factoring Mobile Agents, *Proceedings of the Ninth IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'02).*

Le Goff, H., Stockinger, J.-M., Willers, I., McClatchey, R., Kovacs, Z., Martin, P., Bhatti, N. & Hassan, W. (2001), *Object Serialization and Deserialization using XML. The Compact Muon Solenoid Experiment*, Available at: http://arxiv.org/ftp/physics/papers/0105/0105083.pdf. [Accessed 24/10/2005].

Lee, H. & Shepherdson, J. (2003), A Component Based Multi-agent Architecture to Support Mobile Business Processes, in *Multi-Agent Systems and Applications III: 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003)*, eds. V. Marik, J. Muller & M. Pchouek, Springer-Verlag Berlin Heidelberg, Prague, Czech Republic, 636-636.

Lind, J. (2001), Issues in Agent-Oriented Software Engineering, in P. Ciancarini & M. J. Wooldridge, (eds.), *Agent-Oriented Software Engineering - Proceedings of 1st International Workshop AOSE-2000*, vol. 1957, Springer-Verlag, Berlin, 45-58.

Lind, J. (2002a), *General Concepts of Agents and Multiagent Systems*, Available at: http://www.agentlab.de/general_concepts.html. [Accessed 31/08/2004].

Lind, J. (2002b), Patterns in Agent-Oriented Software Engineering, in F. Giunchiglia, J. Odell & G. Weiß, (eds.), *LNCS:Agent-Oriented Software Engineering III: Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002. Revised Papers and Invited Contributions*, vol. 2583/2003, Publisher: Springer-Verlag GmbH, 47 - 58.

Lingnau, A., Drobnik, O. & Dömel, P. (1995), *An HTTP-Based Infrastructure for Mobile Agents*, Available at: http://citeseer.nj.nec.com/lingnau95httpbased.html. [Accessed 9/03/2000].

Luck, M., McBurney, P. & Preist, C. (2003), *Agent Technology: Enabling Next Generation Computing. A Roadmap for Agent Based Computing,* 1st edn, Agentlink.

Luck, M., Ashri, R. & d'Inverno, M. (2004a), *Agent-based Software Development,* 1st edn, Artech House, Norwood MA.

Luck, M., McBurney, P. & Preist, C. (2004b), A Manifesto for Agent Technology: Towards Next Generation Computing, *Autonomous Agents and Multi-Agent Systems,* 9(3): 203-252.

Luck, M., McBurney, P., Shehory, O., Willmott, S. & Community, A. (2005), *Agent Technology Roadmap*, Available at: http://www.agentlink.org. [Accessed 17/8/2005].

Mangina, E. (2002), *Review of Software Products for Multi-Agent Systems*, AgentLink, http://www.agentlink.org/admin/docs/2002/MAS.pdf.

Manvi, S. S. & Venkataram, P. (2004), Applications of agent technology in communications: a review, *Computer Communications,* 27: 1493-1508.

Marques, P. J., Silva, L. M. & Silva, J. G. (2000), *Going Beyond Mobile Agent Platforms: Component-based development of Mobile Agent Systems*, Available at: http://ede.dei.uc.pt/~pmarques/papers/marques2000e_sea2000.pdf. [Accessed 23/10/2003].

Master Consortium, Available at: http://modeldrivenarchitecture.esi.es/mda_consortium.html. [Accessed 2/12/2005].

Mentz, J. C. (2004), *The Modeling, Design and Implementation of Mobile Agent Systems*, Master Technologiae: Information Technology, Tshwane University of Technology, Pretoria.

Mentz, J. C. & Bierman, E. (2004), *Project TUTA*, Interview in November 2004, Pretoria.

Merriam-Webster Online Dictionary (2004), Available at: http://www.m-w.com/. [Accessed 14/10/2004].

Milojicic, D. (1999), Trend Wars Mobile Agent Applications, *IEEE Concurrency,* 7(3): 80-90.

Milojicic, D., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, B., Kosaka, K., Lange, D., Ono, K., Oshima, M., Tham, C., Virdhagriswaran, S. & White, J. (1998), MASIF: The OMG Mobile Agent System Interoperability Facility, *Personal Technologies,* 2(2): 117-128.

Milojicic, D., Douglis, F. & Wheeler, R. (eds.) (1999), *Mobility Processes, Computers and Agents*, Addison-Wesley, Reading, MA.

Minsky, Y., Van Renesse, R., Sheider, F. & Stoller, S. (1996), Cryptographic support for fault-tolerant distributed computing., in *Seventh ACM SIGOPS European Workshop*, Connemara, Ireland, 109-114.

Mobile Agents for Telecommunications Applications Workshops (2002), Available at: http://dmag.upf.es/mata-ma2002/. [Accessed 25/5/2005].

Müller, J. P. (1998), The Right Agent (Architecture) to do the Right Thing, in *Intelligent agents V : agent theories, architectures, and languages : 5th International Workshop, ATAL'98, Paris, France, July 4-7, 1998 : proceedings*, eds. J. P. Müller, M. P. Singh & A. S. Rao, Berlin ; New York : Springer, Paris, France, 211-225.

Multi-agent Systemen (2004), Available at: http://www.cs.uu.nl/education/vak.php?vak=INFOMAS&jaar=2004. [Accessed 4/05/2005].

Mylopoulos, J., Kolp, M. & Castro, J. (2001), UML for Agent-Oriented Software Development: The Tropos Proposal, in M. Gogolla & C. Kobryn, (eds.), *UML 2001 - The Unified Modeling*

Language, Modeling Languages, Concepts, and Tools, 4th International Conference, vol. 2185 of LNCS, Springer, New York, 422-441.

Ndwana, H. S. & Ndumu, D. T. (1996), An Introduction to Agent Technology, *BT Technol J,* 14(4): 55-67.

Neches, R., Fikes, R., Finin, R. E., Gruber, R., Patil, R., Senator, T. & Swartout, W. R. (1991), Enabling Technology for Knowledge Sharing, *AI Magazine,* 12(1001): 36-56.

Object Serialization Specification (2003), Available at: http://www.waterken.com/dev/Web/Obect/. [Accessed 24/10/2005].

Object-oriented programming and the Objective-C Language (1996), Available at: http://toodarkpark.org/computers/objc/objctoc.html. [Accessed 14/10/2004].

Odell, J. (2002), Objects and Agents Compared, *Journal of Object Technology,* 1(1): 41-53.

OMG Agent Working Group, Available at: http://www.objs.com/agent. [Accessed 20/7/2005].

Omicini, A. (2001), SODA: Societies and Infrastructures in the Design and Analysis of Agent-Based Systems, in P. Ciancarini & M. Wooldridge, (eds.), *Agent-Oriented Software Engineering*, vol. 1975 of LNCS, Springer, New York, 185-193.

Padgham, L. & Winikoff, M. (2003), Prometheus: A Methodology for Developing Intelligent Agents, in F. Giunchiglia, J. Odell & G. Weiss, (eds.), *Agent-Oriented Software Engineering III*, vol. 2585 of LNCS, Springer, New York, 174-185.

Parunak, H. V. D. (2000), Agents in Overalls: Experiences and Issues in the Development and Deployment of Industrial Agent-Based Systems, *International Journal of Cooperative Information Systems,* 3(9): 209-227.

Pavlou, G. (2000), *The Grasshopper Mobile Agent Platform*, Available at: http://www.ee.surrey.ac.uk/CSSR/ACTS/Miami/grasshopper.html. [Accessed 12/10/2005].

Perdikeas, M. K., Chatzipapadopoulos, F. G., Venieris, I. S. & Marino, G. (1999), Mobile agent standards and available platforms, *Computer Networks,* 31: 1999-2016.

Pham, V. A. & Karmouch, A. (1998), Mobile Software Agents: An Overview, *IEEE Communications Magazine*(July 1998): 1-12.

Picco, G. P. (2001), Mobile agents: an introduction, *Microprocessors and Microsystems,* 2(2): 65-74.

Pinsdorf, U. & Roth, V. (2002), Mobile Agent Interoperability Patterns and Practice, in *Ninth IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, Sweden.

Platon, E., Sabouret, N. & Honiden, S. (2004), *T-compound: An Agent-Specific Design Pattern*, Available at: http://www.open.org.au/Conferences/oopsla2004/PaperAO/. [Accessed 20/6/2005].

Procopio, M. (2002), *Object Serialisation and Deserialisation in C#*, Available at: http://www.devhood.com/tutorials/tutorial_details.aspx?tutorial_id=448. [Accessed 24/10/2005].

Recursion Software - Intelligent Mobile Agent Technology (2004), Available at: http://www.recursionsw.com/mobile_agents.htm. [Accessed 11/5/2005].

Report on Workshop 14 at OOPSLA2002: Agent-Oriented Methodologies (2002), Available at: http://www.open.org.au/Conferences/oopsla2002/. [Accessed 16/6/2005].

Roth, V. (1998), Secure Recording of Itineraries through Cooperating Agents, in *ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems:Secure Internet Mobile Computations*, Brussels, Belgium, 147-154.

Roth, V. (2004), Obstacles to the Adoption of Mobile Agents, in *Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM'04)*, Los Alamitos, Calif.,: IEEE Computer Society, Berkeley, California, 296-297.

Roth, V., Braun, P. & Rossak, W. (2002), Conference Session and Workshop on Performance, Interoperability and Applications of Mobile Agent Systems, in *Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'02)*.

Rothermel, K. & Schwehm, M. (1998), Mobile Agents, in A. Kent & J. G. Williams, (eds.), *Encyclopedia for Computer Science and Technology*, M. Dekker Inc., New York.

Saleh, K. & Elmorr, C. (2004), M-UML: an extension to UML for the modeling of mobile agent-based software systems, *Information and Software Technology,* 46: 219-227.

Samaras, G. (2004), Mobile Agents: What about Them? Did They Deliver what They Promised? Are They Here to Stay?, in *Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM'04)*, Los Alamitos, Calif.: IEEE Computer Society, Berkeley, Calif, 294-295.

Sander, T. & Tschudin, C. (1998), Protecting Mobile Agents Against Malicious Hosts, in *Lecture Notes in Computer Science*, vol. 1419, Springer-Verlag, 44-60.

Satoh, I. (2004), Linking Physical Worlds to Logical Worlds with Mobile Agents, in *Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM'04)*, 332-345.

Satoh, I. (2005), Mobile Applications in Ubiquitous Computing Environments, *IEICE Transactions on Communications,* E88B(3): 1026-1033.

Schneider, F. B. (1997), Towards Fault-Tolerant and Secure Agentry, in *11th international Workshop on Distributed Algorithms*, Saarbrücken, Germany.

Schoeman, M. & Cloete, E. (2003), Architectural Components for the Efficient Design of Mobile Agents, in *IT Research in Developing Countries, SAICSIT 2003*, eds. J. Eloff, A. Engelbrecht, P. Kotze & M. Eloff, SAICSIT, Fourways, South-Africa, 48-58.

Schoeman, M. & Cloete, E. (2004), Design Concepts for Mobile Agents, *South African Computer Journal* (32): 13-24.

Schreiber, A. T., Wielinga, B. J., Akkermans, J. M. & Van de Velde, W. (1994), CommonKADS: A Comprehensive Methodology for KBS Development, *IEEE Expert,* 9(3): 28-37.

Sebesta, R. W. (1999), *Concepts of Programming Languages,* 4th edn, Addison-Wesley.

Shih, T. K. (2001), Mobile agent evolution computing, *Information Sciences,* 137: 53-73.

Shoham, Y. (1993), Agent-Oriented Programming, *Artificial Intelligence,* 60: 51-92.

Silva, A. & Delgado, J. (1998), The Agent Pattern for Mobile Agent Systems, in *European Conference on Pattern Langugaes of Programming and Computing*, Bad Irsee, Germany.

Singh, A. K., Sankar, R. & Jamwal, V. (2002), *Design Patterns for Mobile Agent Applications*, Available at: http://autonomousagents.org/ubiquitousagents/papers/papers/22/pdf. [Accessed 24/10/2005].

Sudeikat, J., Braubach, L., Pokahr, A. & Lamersdorf, W. (2004), Evaluation of Agent-Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform, in *The Fifth International Workshop on Agent-Oriented Software Systems (AOSE-2004)*, Columbia University New York City, New York.

Sunsted, T. (1998), *Agents on the Move*, Available at: http://www.javaworld.com/javaworld/jw-07-1998/jw-07-howto.html. [Accessed 5/5/2005].

The TACOMA project (Tromsø And COrnell Moving Agents), Available at: http://www.tacoma.cs.uit.no. [Accessed 11/5/2005].

Tahara, Y., Ohsuga, A. & Honiden, S. (1999), Agent System Development Method Based on Agent Patterns, *Proceedings of the 21th International Conference on Software Engineering*: 356-367.

Taniar, D. (2005), Editorial, *Journal of Mobile Information Systems,* 1(1): 1-2.

Thomas, S. R. (1995), The PLACA Agent Programming Language, in M. Wooldridge & N. Jennings, (eds.), *Intelligent Agents, ECAI-94 Workshop on Agent Theories, Architectures and Languages*, vol. 890 of LNCS, Springer, New York, 355-370.

Tozicka, J. (2003), Airports for Agents: An Open MAS Infrastructure for Mobile Agents, in *Multi-Agent Systems and Applications III: 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, eds. V. Marik, J. Muller & M. Pchouek, Springer-Verlag Berlin Heidelberg, Prague, Czech Republic, 373-382.

Transport Layer Security, Available at: http://en.wikipedia.org/wiki/Secure_Sockets_Layer. [Accessed 1/11/2005].

Travers, M. D. (1996), *Programming with Agents: New metaphors for thinking about computation*, Doctor of Philosophy, Massachusetts Institute of Technology.

Tripathi, A. R., Ahmed, T. & Karnik, N. M. (2001), Experiences and future challenges in mobile agent programming, *Microprocessors and Microsystems,* 25: 121-129.

Tripathi, A. R., Karnik, N. M., Ahmed, T., Singh, R. D., Prakash, A., Kakani, V., Vora, M. K. & Pathak, M. (2002), Design of the Ajanta system for mobile agent programming, *Journal of Systems and Software,* 62(2): 123-140.

Tveit, A. (2001), *A Survey of Agent-Oriented Software Engineering*, Available at: http://www.abiody.com/jfipa/publications/AgentOrientedSoftwareEngineering/. [Accessed 03/08/2004].

Usability (2005), Available at: http://en.wikipedia.org/wiki/Usability. [Accessed 18/11/2005].

Van Aart, C. (2005), *Organizational Principles for Multi-Agent Architectures*, Birkhäuser Verlag, Basel, Switzerland.

Vecchiola, A., Gozzi, M. & Boccalatte, A. (2003), An Agent Oriented Programming Language targeting the Microsoft Common Language Runtime., in *1st International Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Computer Vision and Distributed Computing.*, UNION Agency - Science Press, Plzen, Czech Republic, 6.

Venners, B. (1997), *Under the Hood: The architecture of aglets*, Available at: http://www.javaworld.com/jw-04-1997/jw-04-hood_p.html. [Accessed 24/10/2005].

Venter, L. & Barrow, J. (2003), Can Objects communicate, and why would they want to?, in *South-African Computer Lecturers Association (SACLA 2003)*, Pilanesberg, South-Africa, 5.

Vigna, G. (1997), Protecting Mobile Agents through Tracing, in *3rd ECOOP Workshop on Mobile Object Systems*, Jyväskylä, Finland.

Vigna, G. (2004), Mobile Agents: Ten Reasons For Failure, in *2004 IEEE International Conference on Mobile Data Management (MDM'04)*, Los Alamitos, Calif.,: IEEE Computer Society, Berkeley, California.

Vogler, H., Kunkelmann, T. & Moschgath, M. L. (1997), Distributed Transaction Processing as a Reliability Concept for Mobile Agents, *Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '97).*

Voss, G. (1991), *Object-Oriented Programming An Introduction,* 1st edn, Osborne McGraw-Hill.

Voyager's Intelligent Mobile Agent Technology (2004-2005), Available at: http://www.recursionsw.com/mobile_agents.htm. [Accessed 8/2/2005].

Wang, A. I., Hanssen, A. A. & Nymoen, B. S. (2000), Design Principles For A Mobile, Multi-Agent Architecture For Cooperative Software Engineering.

Web Services, Available at: http://www.w3.org/TR/ws-arch. [Accessed 15/8/2004].

Weyns, D., Helleboogh, A., Steegmans, E., Wolf, T. D., Mertens, K., Boucke, N. & Holvoet, T. (2004), *Agents are not part of the problem, agents can solve the problem*, Available at: http://www.open.org.au/Conferences/oopsla2004/PaperAO/. [Accessed 20/6/2005].

White, J. (1996), *Mobile Agents White Paper*, Available at: http://citeseer.ist.psu.edu/white96mobile.html. [Accessed 7/5/2005].

White, J. E. (1994), *Telescript Technology: The Foundation for the Electronic Marketplace.*, General Magic, Inc., Mountain View, CA, USA.

White, J. E. (1997), Mobile Agents, in J. M. Bradshaw, (ed.) *Software Agents*, AAAI Press/The MIT Press, Menlo Park, California; Cambridge, Massachusetts; London, England, 437-472.

Whitestein Technologies, Available at: http://www.whitestein.com. [Accessed 3/8/2005].

Whitestein Technologies: Methodology (2005), Available at: http://www.whitestein.com/pages/solutions/meth.html. [Accessed 3/8/2005].

Wong, D., Paciorek, N. & Moore, D. (1999), Java-based mobile agents, *Communications of the ACM,* 42(3): 92-102.

Wong, J., Helmer, G., Naganathan, V., Polavarapu, S., Honovar, V. & Miller, L. (2001), SMART mobile agent facility, *The Journal of Systems and Software,* 56: 9-22.

Wood, M. F. & DeLoach, S. A. A. (2000), An Overview of the Multiagent Systems Engineering Methodology, in P. Ciancarini & M. J. Wooldridge, (eds.), *Proceedings of the First International Workshop on Agent-Oriented Software Engineering*, vol. 1957 of LNCS, Springer, New York, 127-141.

Wooldridge, M. (2002), *An Introduction to Multiagent Systems,* 1st edn, John Wiley & Sons Ltd, Chichester.

Wooldridge, M. & Jennings, N. R. (1995), Intelligent Agents: Theory and Practice., *The Knowledge Engineering Review,* 10(2): 115-152.

Wooldridge, M., Jennings, N. R. & Kinny, D. (2000), The Gaia Methodology for Agent-Oriented Analysis and Design, *Autonomous Agents and Multi-Agent Systems,* 3(3): 285-312.

Wooldridge, M. J. & Jennings, N. R. (1999), SOFTWARE ENGINEERING WITH AGENTS:Pitfalls and Pratfalls, *IEEE INTERNET COMPUTING*(May June 1999): 20 - 27.

Zambonelli, F., Jennings, N. R., Omicini, A. & Wooldridge, M. (2001), Agent-Oriented Software Engineering for Internet Applications, in A. Omicini (ed.) *Coordination of Internet Agents*, Springer-Verlag.

Zambonelli, F. & Omicini, A. (2004), Challenges and Research Directions in Agent-Oriented Software Engineering, *Autonomous Agents and Multi-Agent Systems,* 9(3): 253-283.

Zapf, M., Muller, H. & Geihs, K. (1998), Security Requirements for Mobile Agents in Electronic Markets, in W. Lamersdorf & M. Merz, (eds.), *TREC'98, LNCS 1402*, Springer-Verlag, Berlin Heidelberg, 205-217.

Zaslavsky, A. (2004), Mobile Agents: Can They Assist with Context Awareness?, in *2004 IEEE International Conference on Mobile Data Management (MDM'04)*, Los Alamitos, Calif.,: IEEE Computer Society, Berkeley, California, 304-305.