Using Mobile Agents to Solve the Distributed Buying Problem.

by

Kamil Reddy

Submitted in fulfilment of the academic requirements for the degree of Master of Science in the Department of Computer Science, School of Geological and Computer Sciences, University of Natal, Durban.

The financial assistance of the Department of Labour (DoL) towards this research is hereby acknowledged. Opinions expressed & conclusions arrived at, are those of the author and are not necessarily to be attributed to the DoL

Abstract

This study deals with the Distributed Buying Problem, that is, the problem faced by geographically distributed businesses when it comes to optimising buyer time and global businesses resources. It adopts a software agent-based approach to the problem. A literature survey was carried out to review the relatively new field of software agents and mobile agents in particular. The role of agents in electronic commerce was also studied. A mobile agent system was then designed and implemented to serve as a proof-of-concept system for an agent-based solution to the problem. The design and implementation are then discussed.

Preface

The research on which this thesis is based was carried out in the Department of Computer Science, School of Geological and Computer Sciences, University of Natal, Durban, under the supervision of Professor Wayne Goddard.

This thesis represents original work done by the author. Where use has been made of the work of others, it has been duly acknowledged in the text.

Acknowledgements

I would like to express my thanks to:

My supervisor, Professor Wayne Goddard, for his help, guidance and effort in producing this work – especially for sponsoring my trip to the 4th European Agent System Summer School in Bologna, Italy.

My co-supervisor, Dr Johnson Kinyua, for his assistance and guidance during my research.

My friends and my fellow post-graduate students for their support and help – especially Sumeshree Govender, Mark Lewis, Theo Naicker and Deshen Moodley.

Aroon Patel, Johnny du Plessis and Wayne Cronk of Hillside Aluminium for their help in providing me with information on BHP Billiton's Southern African aluminium-smelting operation.

Sören Greenwood for his tireless assistance.

The National Research Fund and the Department of Labour for the funding they have provided.

Table of Contents

List of Figures

1 Introduction

The idea of the software agent has its roots in the notion of the robot as a mechanical device that does work on behalf of people [Bradshaw, 1997]. Software agents can be considered the software equivalent of robots in the physical world – programs that are capable of assisting people by allowing people to delegate work to them. They are software objects that are reactive, autonomous, goal-driven and temporally continuous. Delegating a job to an autonomous program is a concept that makes agent-oriented programming intuitive for the programmer. Mobile agents are software agents that have the additional ability to move from host to host within a network. They are also generally written to execute independently of the host operating system. This allows them to be used to create distributed systems in large networks such as the Internet where multiple operating systems exist. The distributed systems created by mobile agents can have flexible architectures when compared to those created using the client-server paradigm, making them suitable for solving many problems in E-Commerce.

A problem that is as old as commerce itself is that of selecting the best supplier for a purchase. In large businesses this problem is the domain of the purchasing professional or buyer. Buyers today are faced with a multitude of potentially globally distributed suppliers making selecting the best supplier a time consuming task. The task becomes even more difficult in the case of the distributed business enterprise that consists of many self-sufficient business units distributed over a large geographic area. In this case the buyers at each unit cannot act in the interest of the enterprise as a whole unless there is a means to communicate between individual business units. Traditional means such as telephone calls and facsimiles take up too much of the buyer's time. We term this problem of optimising the buyer's time and enterprise costs the Distributed Buying Problem.

1.1 Objective

In this thesis we examine software agents and mobile agents in particular, and their possible use in solving problems in E-Commerce. As an example, we consider the Distributed Buying Problem, and design and implement a prototype for its solution.

We use BHP Billiton's Southern African aluminium-smelting operation as an example of a distributed enterprise – it consists of two aluminium-smelting plants in Richards Bay, and one in Mozambique. Our prototype system allows the buyers at a plant to delegate to the agents the task of communicating with suppliers and plants for quotations and orders. The automation of these processes by the mobile agent system helps optimise the buyer's time and reduce costs to the enterprise, especially through unnecessary buying of materials available at other plants, thus solving the problem.

1.2 Thesis Outline

The rest of this thesis is organised as follows:

Chapter 2 – Software Agents

This chapter presents the results of a survey of software agent literature. The concept of a software agent is introduced here as well as some important definitions of a software agent that lead to a working definition for this thesis. The chapter ends with a taxonomy of agents that defines some of the different types of agents referred to in the literature.

Chapter 3 – Mobile Agents

Chapter three discusses the mobile agent paradigm and the standards that govern mobile agent software. Formal models of mobile agents are also discussed. A list of the advantages of mobile agents concludes the chapter.

Chapter 4 – Aglets and Other Mobile Agent Systems

The Aglets mobile agent system used in this research is described here along with two other mobile agent systems, namely JADE and Hive.

Chapter 5 – Agents in Electronic Commerce

Chapter five presents an overview of traditional, or non-agent-based forms of electronic commerce, and the results of a literature survey into agent-based electronic commerce. Traditional electronic commerce is discussed first in order to place the role and impact of agents in context. Agent-based electronic commerce is then handled by discussing four classification schemes for agents in electronic commerce.

Chapter 6 – The Distributed Buying Problem

In this chapter the Distributed Buying Problem is defined. The role of the buyer in business is discussed, as well as the types of materials buyers deal with and the traditional methods they use. The buying procedure followed at Billiton's aluminium-smelting plants is described, followed by a definition and description of procurement reengineering. The chapter is concluded by discussing how mobile agents can be used for procurement reengineering at Billiton.

Chapter 7 – Design of the Mobile Agent System

The overall design of the mobile agent system, as well as the role of each agent in the system is presented here.

Chapter 8 – Implementation of the Mobile Agent System

Chapter eight discusses the implementation details of the mobile agent system. The hardware and software used in the implementation is listed. The system dataflow is described here to show how data moves through the system. Each agent is then discussed, followed by a discussion of the agent implementation.

2 Software Agents

2.1 Introduction

Though the field of software agent research is relatively new, it has recently seen a surge of activity. Born out of the field of artificial intelligence (AI), agent theories are now implementable largely due to the advent of object-oriented technologies, and the focus on network programming that the Internet has brought about.

There are many perceptions about what constitutes an agent. A survey of the literature will yield the "cliché that there is no one, universally accepted definition of intelligent agent technology" [Nwana & Wooldridge, 1996]. The multitude of definitions exist since there are many authors working in various fields and to different ends, each espousing an idea of agency that brings the definition closer to their field, and work in particular [Franklin & Graesser, 1996]. Implicit in this is that agent research encompasses work done in a wide range of disciplines (for example, distributed artificial intelligence and human-computer interfaces). Researchers can easily mould the definition because the agent metaphor is flexible enough to be applied to most contexts.

To compound this, the popular press has created the impression that software or intelligent agents are poised to change the world of computing, as we know it today. Software companies have latched onto this hype and have begun to label virtually anything an 'agent'.

A few authors (Nwana, Wooldridge and Sycara amongst others) have reviewed the field and have created taxonomies of agents in an attempt to create some uniformity. Agents by different definitions usually find a home in some part of these classification schemes. This in its own does not solve the problem, because one still has to agree with the author's method, or the philosophy behind classifying agents. It does, however, allow one to refer to a particular type of agent with the knowledge that those familiar with the taxonomy will understand.

Another approach that does not offer a strict, single definition of agency is the two-pronged method adopted by Bradshaw [1997]. Like van de Velde he makes a distinction between the ascribing intelligence to an agent, and describing the behaviour of a software agent [Van de Velde, 1995].

Other authors such as Franklin and Graesser [1996] try to avoid 'tight' definitions of agency. They argue that the "only concepts that yield sharp edge categories are mathematical concepts, and they succeed only because they are content free." Instead they offer a loose definition that includes most agents, but which, by their admission, will fail in extreme cases.

2.2 Towards a Definition

The word 'agent', in the context of a software agent, is not far removed from the common meaning of the word agent as "a person who acts for or represents another" [Cambridge Dictionaries Online]. Most, but not all researchers, will agree that in its most basic form, the 'software' agent is a piece of software that represents a user, or another software agent.

Wooldridge and Jennings conducted a seminal survey of the field of agent research. Later they attempted to extract a definition of a software agent from their survey of the literature. They put forward the concept of two separate notions of agency: the first a weak notion, that is relatively widely accepted; and the second, a strong notion, that is "potentially more contentious".

The weak notion states that an agent displays the following properties:

- *Autonomy*: agents operate without the direct intervention of humans or others, and have some control over their actions and internal state.
- *Social ability*: agents interact with other agents (and possibly humans) via an agent-communication language.
- *Reactivity*: agents perceive their environment (which may be the physical world, a user, a collection of other agents, the Internet, or perhaps all of these combined), and respond to changes in it;
- *Pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.

[Wooldridge & Jennings, 1995]

From the many definitions of agents in the literature, it is clear that the weak notion of agency comprises many of the different attributes given to agents by different authors. Examples given later will show that few authors include all the aspects.

The strong notion of agency is more a product of the arguments of AI researchers. They contend that agents should additionally demonstrate the more human-like, or mentalistic qualities of knowledge, belief, intention and obligation, or even emotional states.

2.2.1 Some Example Definitions

Maes provides this definition of an agent:

"Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed." [Maes, 1995]

The software agents of Maes have the condition that they must inhabit an environment that is necessarily complex and dynamic. A dynamic environment is a feature of many definitions from researchers with an interest in AI. This is because if an agent is to function adequately in a dynamic environment, it will need some measure of AI. What makes an environment dynamic or complex is debatable. Furthermore, it is worth noting that she does not include social ability in her definition.

The definition by Russell and Norvig shows that even the fundamental criterion of an agent being a computational system is not universally accepted.

"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors." [Russell & Norvig, 1995].

This definition is wide enough to include most living things, although their text goes on to specify a software agent as having "encoded bit strings as its percepts and actions". Unlike most other definitions that arise from an AI perspective, Russell and Norvig do not specify that the environment must be dynamic. In fact, without restricting the environment or further specifying what is meant by sensing and acting, there is little that differentiates this view of a software agent from any other computer program [Franklin & Graesser, 1996].

Genesereth and Ketchpel take a much more radical approach in their definition based simply on social ability. Their argument here is for standard agent communication languages that will enable multiple agent systems or MASs. A standard language would provide a layer of abstraction between interface and implementation [Genesereth & Ketchpel, 1994] and enable different agents to communicate with each other in order to achieve an objective. Their definition follows:

"...software components that communicate with their peers by expressing messages in an expressive agent communication language." [Genesereth & Ketchpel, 1994]

By this definition one could convert almost any program that transfers information over a network into an agent by forcing it to use an agent communication language for the transfer. It does, however, emphasize the importance some people place on certain aspects of agenthood to the exclusion of others, even within the context of the weak notion.

The next definition is from a proponent of the strong notion of agency. Shoham was the first to propose the paradigm of agent-oriented programming. He describes an agent as:

"An agent is an entity whose state is viewed as consisting of mental components such as belief, capabilities, choices and commitments. These components are defined in a precise fashion, and stand in rough correspondence to their common sense counterparts." [Shoham, 1993]

He goes further to state "What makes any hardware or software component an agent is precisely the fact that one has chosen to analyze and control it in these terms." (This corresponds to Dennett's 'Intentional stance' on observing/predicting behaviour and intelligence. See [Dennett, 1987].) Therefore, whether something qualifies as being an agent or not, is ultimately up to the individual to decide. In order to avoid giving carte blanche to people who would label anything an agent, Shoham does require conditions to be met for giving mental qualities to entities. These are:

- A precise *theory* regarding the particular mental category; the theory must have clear semantics, and should correspond to the common-sense use of the term;
- A *demonstration* that the component of the machine obeys the theory;
- A demonstration that the formal theory plays a *nontrivial role* in analysing or designing the machine; [Shoham, 1993]

Shoham realised the *software* agent under this definition by using a notational language to denote the mental components, logics to dictate the behaviour of each component and finally the creation of a programming language (AGENT-O) based on the former.

2.2.2 The Stereoscopic View

Another approach to defining agents is what we call the stereoscopic view. This states that there are two ways to look at agents. The first is to ascribe agency to the agent as a function of the observer, while the second is to describe an agent as a function of its own behaviour and attributes.

The first view, or agents as an ascription as Bradshaw calls it, is also based on Dennett's intentional stance. Shoham's definition of an agent also falls into this category. To show the argument for this view, consider the following example: If one were to visit a robotics competition, one would see many different looking objects that one would intuitively recognise as robots. These robots are likely made of different components in different configurations, yet one does not need to open them up to determine that they are robots. Indeed, one does not care how they are made, so long as they look and act like robots. In other words their 'robot-ness' is in the eye of the observer. Similarly when looking at agents we see a resemblance that "cannot have to do with similarity in the details of implementation, architecture, or theory, it must be to a degree a function of the eye of the beholder." [Bradshaw, 1997] In other words, as an observer, one ascribes agency to an agent.

The second view, or agents by description, is basically the same as deciding if something is an agent by using the two notions of agency: examine the behaviour and attributes of the entity in question, see if it matches some authoritative description of an agent, and then proclaim it an agent or not. Using the robot analogy, this amounts to opening up each robot, checking the type and configuration of the components to see if they meet the criteria for a robot, and then deciding.

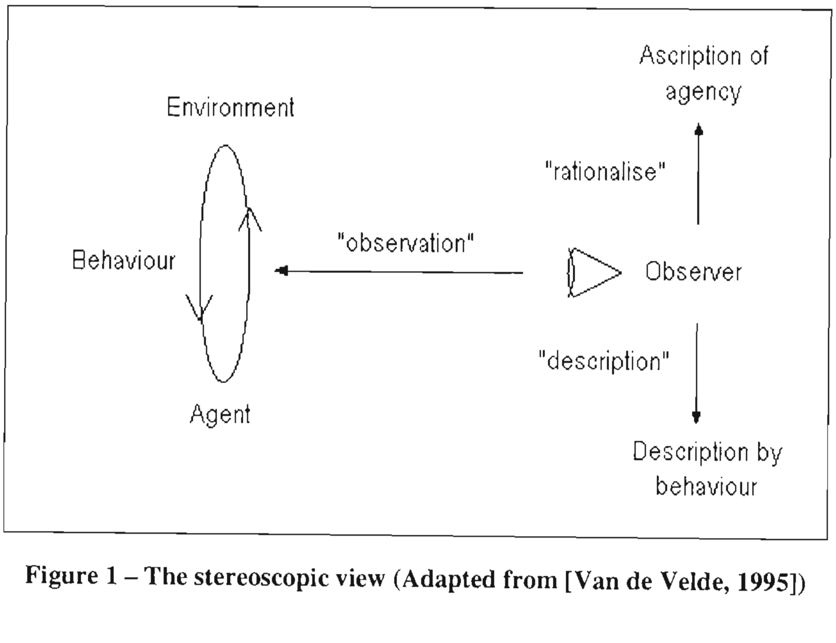


Figure 1 – The stereoscopic view (Adapted from [Van de Velde, 1995])

2.2.3 A Working Definition

The working definition of a software agent for this research is derived from the stereoscopic view. The definitions are from Lange and Oshima [1998].

Agent (End-User Perspective)

An agent is a program that assists people and acts on their behalf. Agents function by allowing people to delegate work to them.

If, from the end-user's perspective, a program appears to assist him and act on his behalf, and he is capable of delegating work to it, he may ascribe agency to it.

Agent (System Perspective)

An agent is a software object that

- *Is situated within an execution environment*
- *Possesses the following mandatory properties:*
  - *Reactive: senses changes in the environment and acts according to those changes*
  - *Autonomous: has control over its own actions*
  - *Goal-driven: is proactive*
  - *Temporally continuous: is continuously executing*
- *And may posses any of the following orthogonal properties:*
  - *Communicative: able to communicate with other agents*
  - *Mobile: can travel from one host to another*
  - *Learning: adapts in accordance with previous experience*
  - *Believable: appears believable to the end user*

The system perspective bears close resemblance to the weak notion of agency, except that: it adds learning and believability, found in the strong notion, as orthogonal properties; and communication is not considered mandatory. The ability to communicate can be considered vital to most agents, but is not necessary for a whole class of agents called Interface Agents (discussed later). Thus, for the sake of completeness it is preferable to have communication as an orthogonal property. On the whole, the system perspective can then be seen to encompass both of Wooldridge and Jennings' notions.

2.3 A Taxonomy of Agents

The lack of a universal agent definition has prompted researchers to create new, more specific definitions in the hope of providing more clarity. Hence the terms such as 'task agent', 'interface agent', 'information agent' etc. Some researchers have propsed schemes. Two schemes mentioned frequently are given in [Nwana, 1996] and [Franklin & Graesser, 1996].

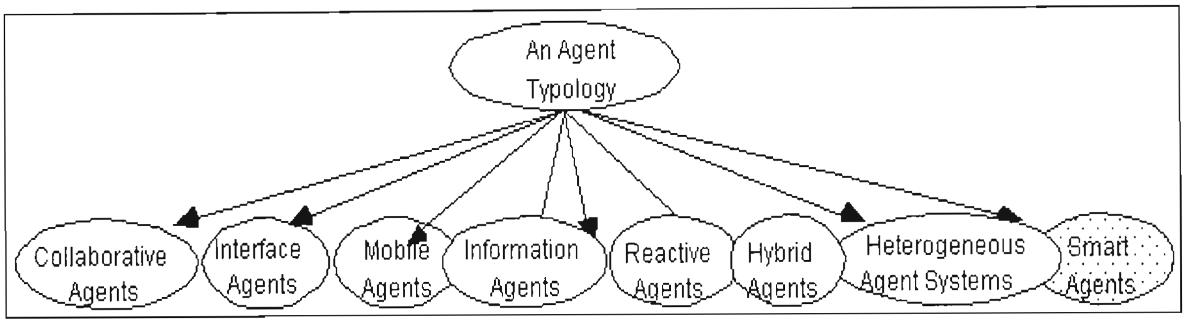


Figure 2 – Nwana's Classification Scheme [Nwana, 1996]

Nwana's scheme is shown above (Figure 2). It is discussed here briefly on an agent-by-agent basis. It was chosen over Franklin and Graesser's as it is more comprehensive.

Collaborative Agents: These are agents whose primary functionality is their ability to cooperate with other agents. The primary reason for having collaborative agents is that as a group, agents may accomplish tasks that they would not be able to accomplish working independently. Most collaborative agents make use of AI techniques that require symbolic representations of their environments. In addition, some form of plan is generally needed to coordinate the activities of the agents in achieving the goal.

Interface Agents: These agents receive user input and deliver results [Sycara et al., 1996]. The trend with interface agents is to mimic a personal assistant that interacts with the user and learns the user's behaviour. The goal is to get away from a direct-manipulation interface and move towards a pro-active interface that does not wait for explicit and detailed instructions from the user. For a debate on direct manipulation vs. interface agents see Maes & Schneiderman [1997].

Mobile Agents: These are agents that are capable of moving from host to host in a network, performing tasks specified by the user, and then returning to the original host. Mobile agents are examined in more detail in the next chapter.

Information or Internet Agents: The primary function of information agents is to make manageable the large amounts of information available from different sources (hence the alternative name of Internet agents). This is achieved by "managing, manipulating or collating" the information. Under Nwana's typology, information agents may also be mobile, or indeed social. A grey area is created here because the distinction between mobile and information, or collaborative and information agents is not perfectly clear. Nwana concedes this, but points out that the distinction should be made in terms of the fact that information agents are defined "by what *they do*", as opposed to the other types that are defined "by what *they are* (i.e. via their attributes...)".

Reactive Agents: Reactive agents do not possess an internal, symbolic representation of their environment. They act by having a specific response to a specific stimulus. In general they are not as complex as those agents employing internal models of their surroundings; however, their complexity lies in the patterns of interactions that are displayed by a group of such agents.

Hybrid Agents: Hybrid agents are agents that are a combination of any two previously mentioned types. They are effective for applications where an agent based on a single design philosophy cannot fully address the problem.

Heterogeneous Agent Systems: These are agent systems that consist of two or more agents from two or more previously mentioned types. For example, a system consisting of a mobile agent, interface agent and a hybrid agent. Agent-based software engineering is largely concerned with enabling heterogeneous agent systems. It is hoped that agents designed to operate on their own in a particular problem domain, may add value to a system of agents that attempt to solve a problem that spans multiple domains. Agent communication languages are vital in this regard.

Smart Agents: Nwana argues that there are three key properties that define agents, namely autonomy, learning and cooperation. If a program does not exhibit some measure of all three elements, then it is not an agent. Conversely, if an agent exhibits these properties equally well, it is a truly smart agent. According to the typology, it is an ideal agent that does not yet exist (hence it is shaded in Figure 2).

3 Mobile Agents

3.1 Introduction

It is the ability to transport themselves across nodes in heterogeneous networks that makes mobile agents a natural tool in many applications. They are also relatively easy to implement for developers already skilled in object-oriented programming, since there are many similarities between object-oriented programming and agent-oriented programming [Iglesias et al., 1999]. They are used mostly in one of four areas:

- *Remote Searching and Filtering* – distributed data access, World Wide Web searches, distributed data mining.
- *E-Commerce* – electronic marketplaces and auctions, business-to-business systems.
- *Networking* – dynamic routing (allowing for active networks), network mapping, load balancing.
- *Mobile Computing* – mobile agent execution environments for mobile or intermittently connected devices (Personal Digital Assistants, laptops, cellular phones), communication in wireless and ad-hoc networks, off-line searches.

Mobile agents are not an exclusive solution in any of these application areas. There are alternatives, such as client-server architectures, that can address each area instead. A mobile agent-based architecture, however, can address them all at once [Harrison et al., 1995].

3.2 The Mobile Agent Paradigm

The use of mobile agents is a move away from traditional client-server and code-on-demand techniques in distributed computing. The mobile-agent paradigm allows for greater flexibility in how distributed systems are designed. The three paradigms are discussed here in terms of service, resource, know-how/code, and processor configuration.

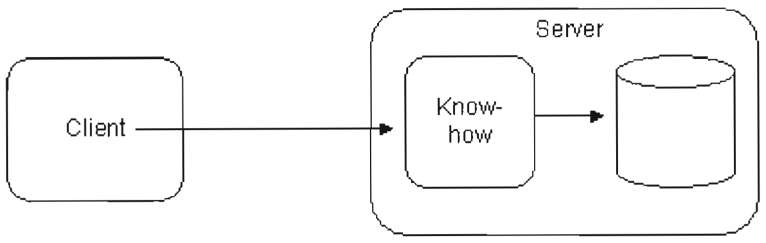


Figure 3 – Client-Server Paradigm [Lange & Oshima, 1998]

In the client-server paradigm, the server advertises a service. The service is usually based on a resource, such as a database, situated on the server. The know-how (code) to query the database and the processor required to run that code are also on the server. Therefore the server holds the resources, code and processor. The Client-Server approach is the most common and oldest approach to network programming. It supports many technologies; Remote Procedure Calling (RPC), Common Object Request Brokering Architecture (CORBA) and Java remote method invocation (RMI) are the most important.

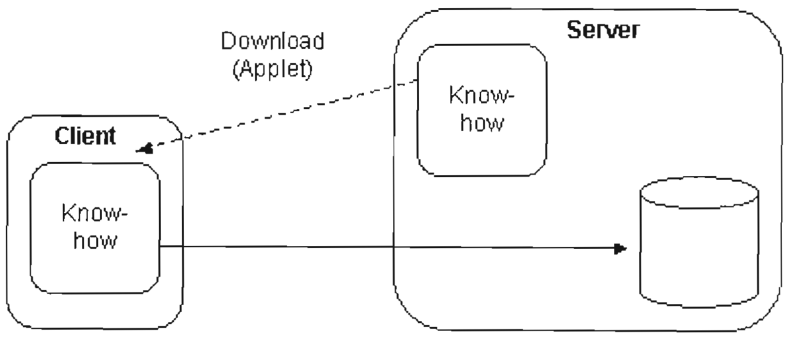


Figure 4 – Code-on-Demand Paradigm [Lange & Oshima, 1998]

The code-on-demand paradigm gives the required resource and processing power to the client. The client, however, does not possess the code or know-how to operate on the resource. This means that the client does not require pre-installed code – the service know-how is downloaded from the server and processed when needed. Java applets are examples of code-on-demand.

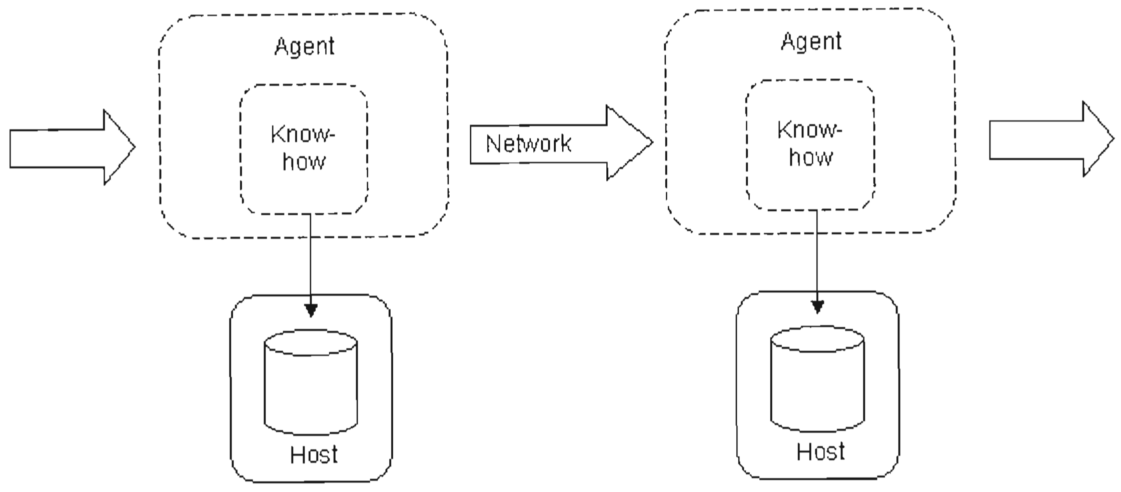


Figure 5 – Mobile Agent Paradigm [Lange & Oshima, 1998]

In the mobile agent paradigm, the host replaces the client and server. Each host offers services based on its resources and processing power. The agent carries the know-how to the appropriate host or hosts and executes it there. Services in client-server and code-on-demand architectures are usually specified at design-time. Mobile agents facilitate flexibility by allowing services to be customized at any time since the know-how is host-independent.

3.3 Standardization and Agent Frameworks

Many proprietary systems have been created. Agents created in one type of system cannot easily migrate to another type of system, nor can they communicate easily with agents of another system. Without these abilities the potential of multi-agent systems cannot be realised. The Object Management Group (OMG) and the Foundation for Intelligent Physical Agents (FIPA) are both involved in defining standards for agents. The Object Management Group adopted the Mobile Agent System Interoperability Facility (MASIF) standard for mobile agents in 1998. It has not changed since then. The latest standard released by FIPA is FIPA2000; however, it is an evolving standard and elements of the specification are constantly being superseded by newer versions. (See http://www.fipa.org/specifications/identifiers.html.)

Prior to 1999, FIPA and the OMG had no formal link and consequently their views on similar issues were divergent at times. There is now a formal link between the OMG's Agent Platform Special Interest Group, and FIPA, to align the thinking of both organisations in order to achieve a common standard. The OMG has released an Agent Technology Green Paper to help identify and build consensus [Odell et al., 2000]. Although the Green Paper is more a discussion document, there are some fundamental differences from the MASIF standard (including the definition of an agent). Nevertheless we will only deal with the MASIF specification.

The OMG MASIF standard and the FIPA standard focus on different aspects of interoperability amongst agents. The MASIF standard aims at standardizing agent platforms, that is, the software execution environments of agents. It focuses more on mobility and enabling the transfer of agents between different platforms. The FIPA standard, on the other hand, focuses on facilitating interoperability through the use of standard agent communication languages. Although there are not many agent systems that currently conform to MASIF [Suri, 2002], the reference models of both standards will be looked at here: MASIF because of its strong emphasis on mobility; and FIPA because of its importance in multi-agent systems and because it is fast becoming the de facto standard. The Aglet agent system used in this thesis conforms to the MASIF specification.

3.3.1 The MASIF Standard

The MASIF standard defines certain key concepts that are used in the reference model. Of those, we discus only those that are important to the general understanding of the mobile agent model. All the definitions listed are from the 1998 MASIF Specification [MASIF'98].

The only two types of agents MASIF considers are stationary and mobile agents:

Stationary Agent

A stationary agent executes only on the system where it begins execution. If the agent needs information that is not on that system, or needs to interact with an agent on a different system, the agent typically uses a communications transport mechanism such as Remote Procedure Calling (RPC).

The standard does not say much more about stationary agents except that: the communications needs of a stationary agent can be met using the same technologies that are used in distributed object systems (e.g. CORBA, DCOM, and RMI); and that stationary agents can be part of mobile agent systems. Typically, stationary agents serve as agent 'wrappers' for services on the systems they are resident on. That is, they serve as a go-between for other agents that require the service and make the underlying implementation details of the particular service transparent to the other agents.

In the next definition of a mobile agent, the word 'object' is used. This is because the specification describes agent systems in terms of objects [MASIF'98]. As such, the word 'object' may be used interchangeably with the word 'agent' and the definition will still hold. (The definition is more comprehensive than that given earlier in Nwana's taxonomy.)

Mobile Agent

A mobile agent is not bound to the system where it begins execution. It has the unique ability to transport itself from one system in a network to another. The ability to travel permits a mobile agent to move to a destination agent system that contains an object with which the agent wants to interact. Moreover, the agent may utilize the object services of the destination agent system.

Mobile agents are often defined by their ability to suspend their state of execution before migrating to another system, and then resuming it there. The MASIF specification declares that an agent migrates with both its code and its state. Agent state is considered to be the execution state (program counter, stack etc.) as well as any other attributes that enable an agent to resume execution at the destination agent system. This then prompts the definition of an agent system.

Agent System

An agent system is a platform that can create, interpret, execute, transfer and terminate agents. An agent system is uniquely identified by its name and address. A host can contain one or more agent systems.

In order to specify all the functions a MASIF-compliant agent system must implement, an interface is defined using the Interface Description Language. Different types of agent systems are supported by the standard. Agent systems may use different languages, and in turn the languages may employ multiple serialization techniques. The combination of agent system type, language and serialization method refers to the profile of an agent. MASIF is primarily involved in enabling interoperability with agents of the same profile. The final building block of the model is the concept of a 'place'.

Place

When an agent transfers itself, the agent travels between execution environments called places. A place is a context within an agent system in which an agent can execute. This *context can provide functions such as access control. The source place and the destination place can reside on the same agent system, or on different agent systems that support the same agent profile.*

The functions a place offers extend much further than access control. A full set of uniform services should be available to an agent in any place. A place can thus be seen as the operating system from an agent's point of view [Lange & Oshima, 1997]. The entire model may now be seen in the following figure.

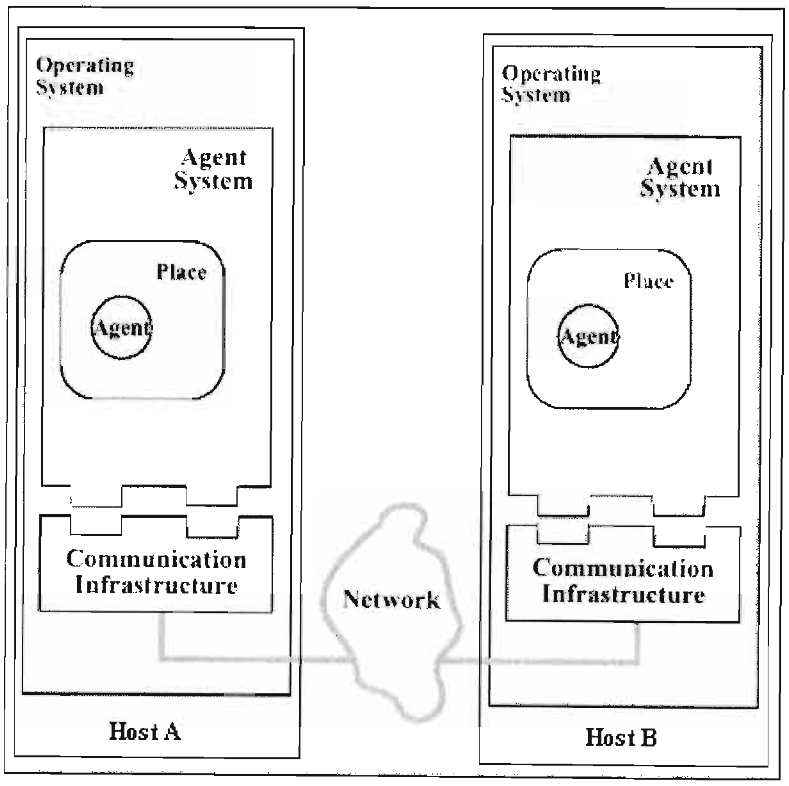


Figure 6 – The MASIF architecture (Adapted from [MASIF98, page 10])

To summarise: An agent resides in a place, possibly with other agents. A place in turn is contained in the agent system. The agent system may have many places (not shown in Figure 6) with different access control policies to discriminate between authorised and unauthorised agents. The agent system relies on the host operating system's resources and facilities.

Access control is mentioned in the definition of a place. Access control forms part of the broader topic of security (which is out of the scope of this work). In the MASIF standard, access control is centred on a system of authorities and authentication. Since an agent ultimately represents some legal entity that is responsible for its actions, an agent authority is defined to identify that entity (see Stuurman & Wijnands [2001] for a look at the legal issues surrounding agents). To avoid agents masquerading as other agents, the specification allows for the authentication of agent authorities. Authorities and authentication are based on the CORBA standard.

Only the architectural components of the MASIF model have been discussed thus far. MASIF also specifies the process of agent transfer from one host to another. The process consists of initiating an agent transfer (dispatch), and receiving an agent transfer. To initiate a transfer, an agent makes an API (Application Programmer Interface) call to the source agent system. The source agent system then notifies the destination agent system. If the destination agent system approves the transfer, the following actions are taken at the source agent system:

Agent Dispatch

1) *Suspend the agent (halt the agent's execution thread)*
2) *Identify the pieces of the agent's state that will be transferred*
3) *Serialize the instance of the Agent class and state*
4) *Encode the serialized agent for the chosen transport protocol*
5) *Authenticate client*
6) *Transfer the agent*

The destination agent system performs these actions to receive the agent:

Agent Reception

1) *Authenticates client*
2) *Decodes the agent*
3) *Deserializes the Agent class and state*
4) *Instantiates the agent*
5) *Restores the agent state*
6) *Resumes agent execution*

To keep track of agents that may be constantly migrating from host to host, an interface called the *MAFFinder* interface is used.

3.3.2 The FIPA Standard

The FIPA standard comprises many specifications with an Abstract Architecture Specification at the highest level, down to a specification for Bit-Efficient Encoding of messages, at the lowest. Figure 7 represents an overview of the FIPA model featuring the key architectural elements. For the purposes of obtaining a broad overview of the standard, only these elements are discussed here.

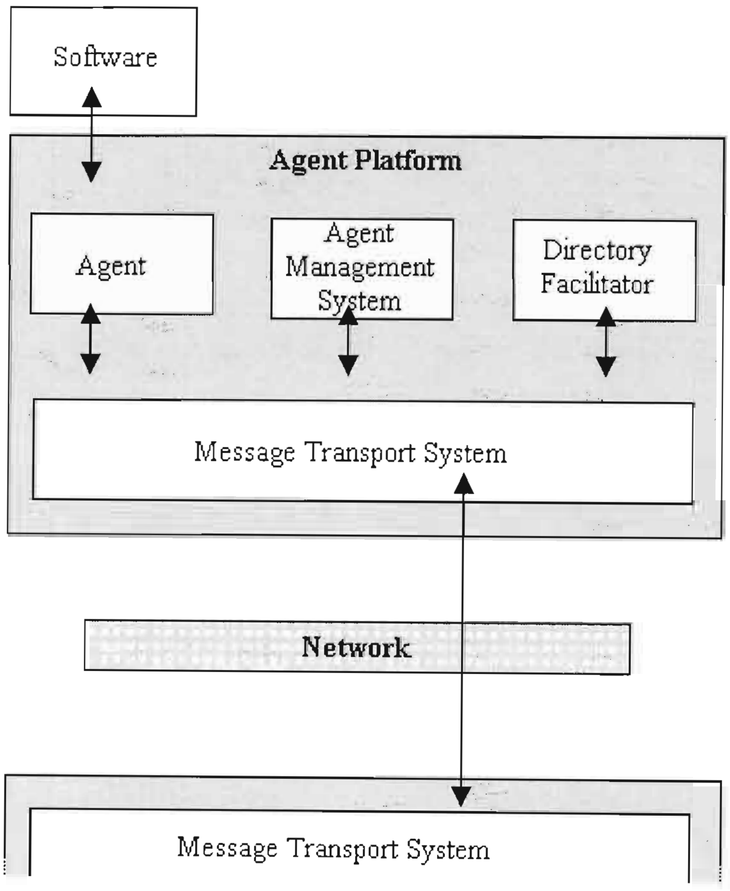


Figure 7 – The FIPA Architecture (Adapted from FIPA XC00023H, page 4)

The Agent Platform is roughly analogous to the Agent System in the MASIF specification. It is defined by FIPA as such:

Agent Platform

An Agent Platform provides the physical infrastructure in which agents can be deployed. The AP consists of the machine(s), operating system, agent support software, FIPA agent management components (Directory Facilitator, Agent Management System and Message Transport Service) and agents. [FIPA XC00023H]

The FIPA agent platform, unlike the MASIF agent system, does not necessarily have to be on one host. A single agent platform may span multiple hosts (this is not shown in the diagram).

FIPA does not distinguish between stationary and mobile agents. It defines an agent as:

Agent

An agent is a computational process that implements the autonomous, communicating functionality of an application. Typically, agents communicate using an Agent Communication Language (ACL). [FIPA XC00001J]

Agents inhabit the agent platform. They provide services such as access to communications facilities and external software. These services combine to form a "unified and integrated execution model" [FIPA XC00023H] for agent-based systems. Agents have an Agent Identifier (AID) to associate them with the legal entities that own them. FIPA allows for an agent to have more than type of name. The naming mechanism provides for nicknames or aliases to be included in the agent identifier. All the agent management components of the AP are implemented as agents.

The component of the agent platform responsible for managing agents is the Agent Management System:

Agent Management System

An Agent Management System (AMS) ... exerts supervisory control over access to and use of the AP. Only one AMS will exist in a single AP. The AMS maintains a directory of AIDs which contain transport addresses (amongst other things) for agents registered with the AP. The AMS offers white pages services to other agents. Each agent must register with an AMS in order to get a valid AID. [FIPA XC00023H]

Transport addresses represent the physical addresses at which agents can be contacted. The white pages services referred to include services to locate agents as well as naming and access control services.

The next component of the agent platform is the Directory Facilitator:

Directory Facilitator

A Directory Facilitator (DF) ... provides yellow page services to other agents. Agents may register their services with the DF or query the DF to find out what services are offered by other agents. Multiple DFs may exist within an AP and may be federated. [FIPA XC00023H]

The use of federated DFs implies that, if a DF is queried for a service that is not contained in its list of services, then it will query the other DFs for that service. The yellow pages services offered are service location and service registration. It is important to note the difference between white page services offered by the AMS and yellow page services offered by the DF. White page services involve finding individual agents while yellow page services involve finding specific services.

The Message Transport Service is one of the most important components of the AP since FIPA bases its interoperability on communication transparency.

Message Transport Service

The Message Transport Service (MTS) provides a mechanism for the transfer of ACL messages between agents. The agents involved may be local to a single AP or on different APs. The MTS is also called the Agent Communication Channel. [FIPA XC00067D]

FIPA does not restrict agents that are local to a single AP, and native to that AP, from using proprietary communication mechanisms. There are three possible ways for agents on different APs to communicate with one another. The first method involves the agent on the local AP sending a message to the local MTS. The local MTS then routes this message to the appropriate remote MTS, which delivers it to the target agent. The second method requires the agent on the local AP to send the message directly to the remote MTS for delivery to the target agent. The third method is a direct communication from the local agent to the remote agent without the use of the MTS. *This method of communication is not supported by FIPA.*

The final element of the AP is software.

Software

Software describes all non-agent, executable collections of instructions accessible through an agent. [FIPA XC00023H]

Software provides agents with a means to update and upgrade their abilities. Agents may acquire new communications, security or negotiation protocols through software.

3.4 Formal Software Models for Mobile Agents

Samaras et al., [1999] propose formal models for distributed systems based on mobile agents. The models they propose are extensions of the traditional client-server model that cater for clients that are mobile (i.e. connected to the network via wireless links). They do however also hold true in the case where the clients are not mobile.

The first model is called the client-agent-server (C/A/S) model. In this model, an agent is situated in the path between the client and the server. The agent acts as a proxy for the server and all the services it offers, or alternatively, there may be multiple agents representing the different services offered by the server. It is also possible to have agents only for certain services. The communication between the client, server and agent is done by messaging and queuing.

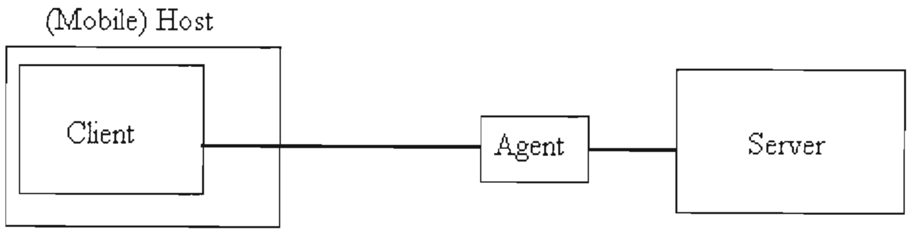


Figure 8 – The Client-Agent-Server Model (Adapted from [Samaras et al., 1999])

The second model is called the client-intercept-server (C/I/S) model. A pair of agents is inserted in the path between the client and the server. The client-side and server-side agents intercept communication emanating from the client and the server respectively. The agents can then use messaging to communicate with each other.

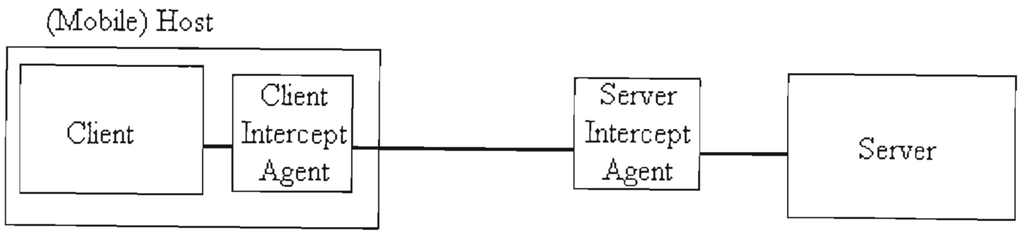


Figure 9 – The Client-Intercept-Server Model (Adapted from [Samaras et al., 1999])

The C/A/S and C/I/S models utilise stationary agents that are bound to the server or both client and server. The mobile agent (MA) model describes a system where an agent is free to move around the network. Communication in the MA model can be either by message passing, or by messenger agents carrying messages from host to host. It is also permissible for transient C/A/S and C/I/S models to be created under the MA model, depending on the configuration of agents, clients, and servers at a given time.

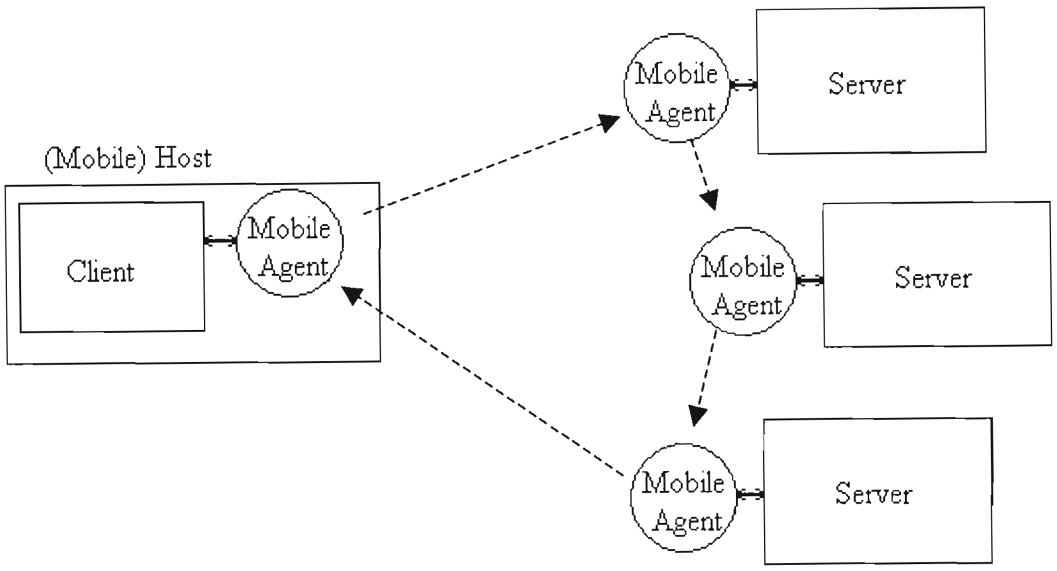


Figure 10 – The Mobile Agent Model (Adapted from [Samaras et al., 1999])

The MA model can be used to extend the client-server, C/A/S and C/I/S models if messenger agents are used to carry information from the client-side to the server-side (instead of message passing). The resultant models are called the 'mobile' client-server, 'mobile' client-agent-server and 'mobile' client-intercept-server models.

3.5 Advantages of Mobile Agents

Distributed object systems are currently the technology of choice in distributed systems programming. These systems are usually based on client-server architectures. Mobile agents hold several advantages over these systems. It should be noted though, that many mobile agent implementations exploit distributed object technology such as CORBA and Java RMI.

Reduction of network traffic: During normal client-server communication, especially where RPC or security protocols are used, there may be many two-way flows of information between client and server. A mobile agent that can migrate to the server enables this communication to take place locally.

Asynchronous and autonomous execution: Mobile agents can operate effectively in networks where connections are frequently broken. An agent's thread of execution is autonomous from the process that created it. Therefore, once an agent is dispatched to a remote host, it may continue to execute there even if the network link to its originating host is broken. When the link is restored the agent may then return to its originating host.

Encapsulation of protocols: In a distributed system environment, the client and server (or source and destination hosts) usually both own the protocol code for receiving and transmitting information. If the protocol changes, it is difficult to upgrade the protocol code [Lange & Oshima, 1998]. Mobile agents avoid this problem by migrating to each host and establishing a communications channel between themselves. This channel may use any protocol, including a proprietary protocol, which the agents support.

Existence in heterogeneous environments: As the size of the network increases in a distributed system, it is more likely that the hosts will differ from one another. Most mobile agents are built with languages, such as Java, that are host-independent [Butte, 2002]. They are governed only by their agent platforms or agent systems. This allows the distributed systems developer to view all hosts in a similar way since a uniform set of services is available at every host.

Robustness and fault-tolerance: The mobile agent's ability to sense its environment and react to changes allows it to be more robust and fault tolerant than traditional systems. For example: If a host is shutting down, it may inform the agents residing on it, which can then dispatch themselves to another host. Alternatively, before the host shuts down agents may create clones of themselves at other hosts.

Efficient remote searching and filtering: In the case of a large remote database, a client would normally perform an SQL query that could return potentially large result sets. A mobile agent could perform this query efficiently by migrating to the database server and performing the query there. Mobile agents are also able to open and search flat text files. Since most of the world's data is in flat text files [Harrison et al., 1995], this is a major advantage.

4 Aglets and Other Mobile Agent Systems

4.1 Introduction

This chapter discusses three software agent packages: Aglets, JADE and Hive. Aglets is discussed in more detail than the others since this research was implemented using Aglets. Aglets is IBM's open-source implementation of mobile software agents, produced under the IBM Public Licence. The JADE agent framework is developed by Telecom Italia Lab (CSELT). It is distributed under the GNU Lesser General Public License and is therefore open-source. Hive is a distributed agent platform created by the MIT Media Lab and is freely available under the GNU General Public License. All the systems are written in Java.

Researching mobile agents has shown that Aglets is one of the most popular mobile agent systems. It is mostly used in academia as a tool for teaching the basics of mobile agents. At the time of choosing a mobile agent system, Aglets was the most popular, freely available system with a large user-base. Since that time the FIPA standard has become the preferred standard and Aglets has become less popular as a research tool. JADE, being FIPA compliant has not had this problem. It is very popular amongst academic and commercial researchers researching mobile and intelligent agents [JADE homepage]. Hive differs from the others because it adopts a different analogy for the distributed system, and it is concerned with mobile agent control of electronic devices. Hive is also owned and written by an academic institution, while companies own Aglets and JADE.

4.2 Aglets

The Aglet Software Development Kit (ASDK) is used to create Aglets. The kit comprises an API, a MASIF-compliant agent system, a proprietary communication protocol, source code and documentation. The aglet model has four basic elements implemented as *objects*: aglet, proxy, context and message.

Aglet: The aglet object corresponds to the MASIF mobile agent. Aglets that do not make use of their mobility may be regarded as MASIF stationary agents. Aglets are considered autonomous because they execute independently of the process that created them, and run their own thread of execution at each host. They are considered reactive because they can respond to events in the agent system and incoming messages.

Proxy: The proxy object acts as a handle to an aglet. All operations on an aglet should be performed using that aglet's proxy. The purpose of a proxy object is to disallow direct access to an aglet's public methods. Proxy objects on a local host may reference an aglet that has migrated to a remote host. Thus, proxies also provide location transparency for the handling of aglets.

Context: The context object implements the MASIF place. Agents execute all their operations in a context. The context offers agents a uniform execution environment.

Message: Aglets use message passing to communicate rather than method invocation. Message objects are passed between aglets to implement this. Message passing allows for synchronous and asynchronous communications. Each aglet has a message manager that handles incoming messages and replies.

The Aglets agent system is implemented through the Tahiti Server. It provides the functionality of an agent system as detailed in the MASIF standard. Tahiti listens on a port for incoming requests for messages and migrating aglets. Whether these messages and migration requests are allowed or denied, depends on whether they satisfy the security criteria specified for the server. Any aglet resident on a Tahiti server may have any of the fundamental aglet operations (discussed later) performed on it by the user.

Aglets use a proprietary communication protocol called the Agent Transfer Protocol (ATP). ATP is modelled on HTTP and is used for the transfer of agents and messages between agent systems

4.2.1 Fundamental Aglet Operations.

The fundamental aglet operations are those that describe the life cycle of an aglet. They are: creation, cloning, dispatching, retraction, deactivation, activation, and disposal.

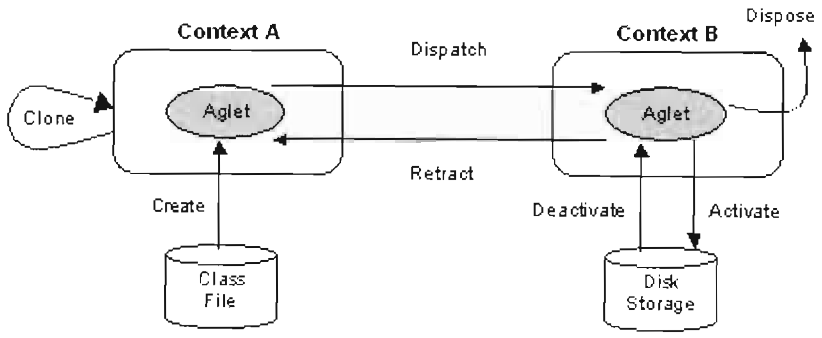


Figure 11 – The Aglet Life Cycle [Lange & Oshima, 1998]

Creation: A context object's *createAglet* method is used to create a new aglet. During creation an aglet is assigned a unique identifier and inserted into the context. It is then initialised and begins execution within the context.

Cloning: Cloning provides an alternative way to instantiate a new aglet. An aglet's proxy object's *clon*e method is used to create a clone of itself. The clone is given a unique identifier at the time of instantiation and begins its own, new thread of execution.

Dispatching: Calling the *dispatch* method of an aglet's proxy object dispatches the aglet from its current context to the destination context. The aglet's thread of execution is halted before a dispatch and a new thread of execution begins when it reaches the destination context.

Retraction: Calling the *retract* method of an aglet's proxy object removes the aglet from its current context and inserts it into the context where the call was made.

Activation and deactivation: The *deactivate* method of an aglet's proxy object causes the aglet to stop its thread of execution and save its state to secondary storage. The *activate* method returns the aglet to its previous state in memory and re-starts the aglet's thread of execution.

Dispose: Calling the *dispose* method of an aglet's proxy object results in the aglet being removed from the context and garbage collected.

The Aglet API is a lightweight API similar to that of the Java Applet (the term Aglet was derived from the joining of 'agent' and 'applet'). It also employs a delegation-based event model, where events are generated for each of the fundamental operations. Listener interfaces are specified that handle theses events. They are the *ContextListener* (creation and disposal), *CloneListener*, *MobilityListener* (dispatching and retraction) and *PersistencyListener* (activation and deactivation) interfaces. Abstract adapter classes (*CloneAdapter*, *MobilityAdapter* etc.) are used to implement the listener interfaces (see Table 1). The tasks that the programmer wishes the aglet to execute are implemented by overriding the methods of the adapter classes. The general behaviour of the aglet, or the tasks that can be executed without events being fired, are specified in the *run* method of the aglet.

Adapter Class	Method to Override
CloneAdapter	*onCloning* – Initialises the newly cloned aglet. *onCloned* – Called in original aglet when the cloning has taken place.
MobilityAdapter	*onDispatching* – Called when an attempt is made to dispatch the aglet. *onArrival* – Initialises the newly arrived aglet. *onReverting* – Called when an attempt is made to retract the aglet from a remote location.
PersistencyAdapter	*onDeactivating* – Called when an attempt is made to deactivate the aglet. *onActivation* – Initialises the newly activated aglet.

Table 1 – Methods that require overriding to specify aglet behaviour.

4.2.2 Aglet Communication

Aglets communicate primarily through message passing. There are three distinct types of messaging that can be used with aglets. The first type of message, simply called *message*, is used for synchronous communication. When a message of type *message* is sent, the aglet's thread of execution blocks until the reply is received. The second type of message is called a *future* or an *async* message. Future messages allow for asynchronous communication because they are non-blocking. A future message can also be configured to wait a specified period of time for a reply, during which it blocks execution. The third type of message is a called a *oneway* message. The oneway message does not require a reply and is therefore also non-blocking. All messages types are sent by making calls to an aglet's proxy. The message is then transferred via ATP.

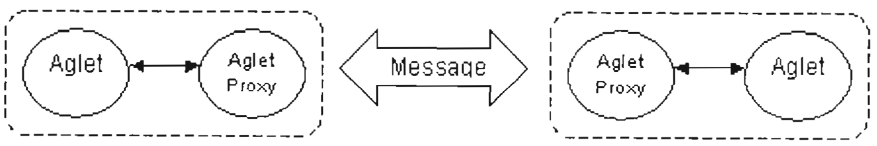


Figure 12 – Aglet Messaging

4.2.3 Aglet Security

This work does not concern itself with the broader issues of agent security. A basic knowledge of the aglets security mechanism is necessary, as it governs how aglets move between, and execute on hosts in the network.

Security is implemented by each Tahiti server in a distributed system. The security settings are specified by a configuration file that uses the Java Development Kit 1.2 policy definition file syntax. Each aglet that executes in a context on the server is granted permissions determining how it may use the various resources of the host computer. Permissions are granted depending on the current context, the code base that the aglet originated from, and who the owner and manufacturer of the aglet are. The owner denotes the legal entity that instantiated and launched the aglet, while the manufacturer is the legal entity that implemented the aglet. Aglet owners and manufacturers may define protections for their aglets. Protections serve to protect an aglet's methods being invoked by unauthorised aglets or programs.

4.3 JADE

The Java Agent Development framework, or JADE, is a software framework for developing multi-agent systems. The JADE package includes a FIPA-compliant agent platform, multiple communication protocols, source code and documentation.

The JADE agent platform is implemented through one or many *containers*. Containers are RMI server objects that manage agents and provide them with an execution environment. They may run on different hosts provided an RMI link can be maintained with the other containers. This is in line with the FIPA specification that an agent platform itself may be distributed amongst multiple hosts in a network. There is a main container that contains the Agent Management System, Directory Facilitator, Agent Communication Channel (or Message Transport Service) and user agents. JADE provides a graphical user interface to the main container that allows the user to view the entire agent platform and manipulate agents within it.

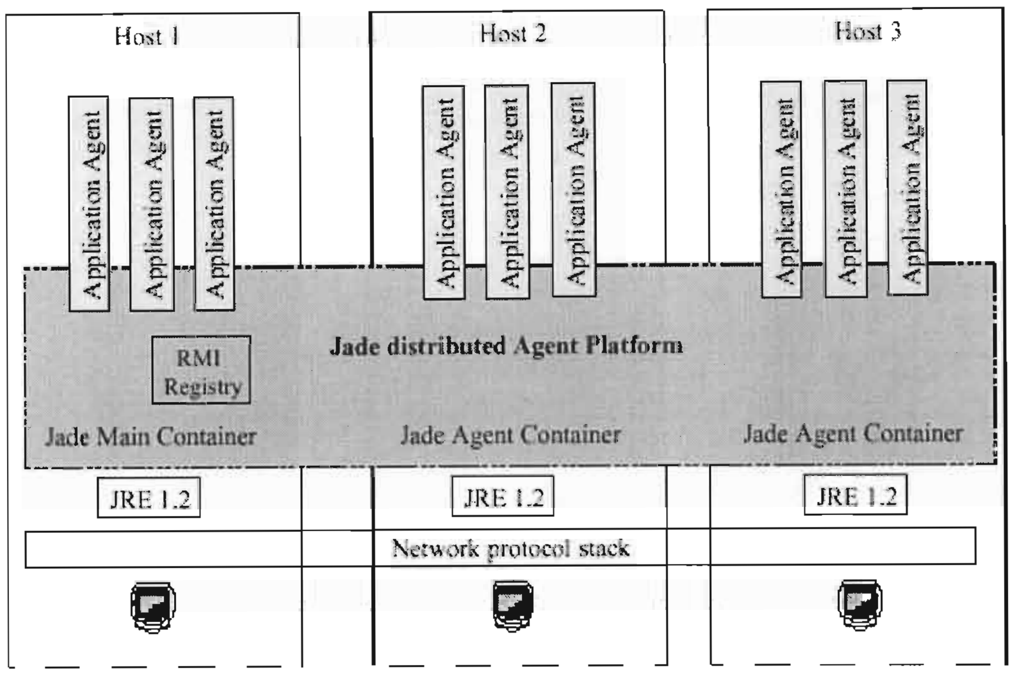


Figure 13 – The JADE Agent Platform [Bellifemine et al., 2002]

JADE agents do not have a single *run* method where arbitrary tasks may be performed. JADE has an abstract *Behaviour* class that must be sub-classed by the user in order to specify the required tasks. Behaviours are added to the agent in a start-up method after instantiation. Some sub-classes of the *Behaviour* class are provided ready to be used. They are child-classes of the *SimpleBehaviour* and *ComplexBehaviour* classes. The *SimpleBehaviour* class allows for single, atomic units of computation that can be executed without interruption, while the *ComplexBehaviour* class allows for more complicated behaviours that may be interrupted. There are two reasons for having these two particular classes: JADE uses a thread-per-agent concurrency model instead of a thread-per-behaviour model; and it employs round robin non-pre-emptive multithreading. This means that when it is an agent's turn to perform a task, all the other agents in its container cannot do anything.

Communication in JADE is done by passing FIPA-ACL message objects. In the case where agents reside in the same container, the message is passed as an event. If agents are on the same agent platform but on different hosts, the message is passed via RMI. Where agents reside on different agent platforms, the message is sent using the Internet Inter-ORB Protocol (IIOP). The latest versions of JADE also support HTTP, Simple Mail Transfer Protocol (SMTP) and Wireless Application Protocol (WAP).

4.4 Hive

The Hive distributed system uses the analogy of an 'ecology of agents', where agents interact with each other to produce the application. Hive agents are designed to facilitate mobile computation in a living or working environment, by providing agent wrappers for electronically controllable devices – in effect, to create distributed mobile agent-based control systems. They are, however, not restricted to this domain and can be used for more common mobile agent applications.

The hive architecture consists of *cells*, *shadows* and *agents*. Agents are located in cells, which provide them with an execution environment, access to communication facilities and local resources. The local resources of each cell are represented by shadows. Shadows serve as an interface between agents and the resources (hardware or software) that the shadows represent. The use of a shadow as an interface allows access to the resource to be limited to the interface methods, and to be governed by a security policy.

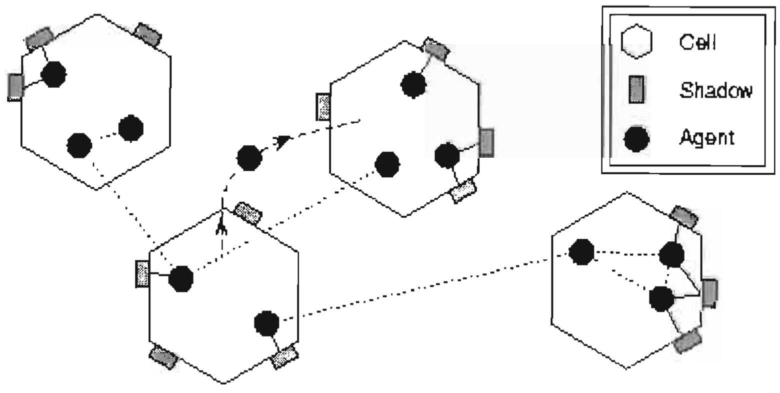


Figure 14 – The Hive architecture [Minar et al., 1999]

Hive utilises a semantic and syntactic lookup scheme to establish and maintain connections between agents and shadows. The syntactic scheme lets agents and shadows find each other based on their Java object type. The semantic scheme allows them to locate each other based on an XML description of their capabilities. Java RMI is used to facilitate communication.

The Hive agent lifecycle is similar to that of an aglet. Table 2 shows a mapping of Hive agent and aglet API calls.

Hive method name	Aglet method name
doLocalSetup	*OnCreation*
doBehaviour	*Run*
onDying	*OnDisposing*
OnMoving	*OnDispatch*
doLocalCleanup	*OnDisposing*

Table 2 – Hive API methods and the corresponding Aglet API methods

5 Agents in Electronic Commerce

5.1 Introduction

Electronic Commerce (E-Commerce) and Electronic Business (E-Business) have many definitions. There is a lack of consensus on the meaning of either term, or indeed, if they are interchangeable. We adopt the view that they are the same, and define the term E-Commerce as the process of buying, selling, or exchanging products, services, or information via computer networks [Turban et al., 2000]. E-Commerce can be seen to consist of one or more of the following (life-cycle) phases: need identification, product brokering, merchant brokering, negotiation, purchase and delivery, and product/service evaluation [Turban et al., 2002]. Agents may be used in E-Commerce either as individual agents that participate in one or more life-cycle phases, or as part of multi-agent systems or frameworks designed to facilitate E-Commerce.

E-Commerce is generally classified according to the kind of transaction, for example business-to-consumer. We find from the literature that research applying software agents to E-Commerce is mainly concerned with following classes of E-Commerce: business-to-business, business-to-consumer, consumer-to-consumer and mobile commerce (E-Commerce that makes use of wireless mobile devices such as cell-phones and PDAs). Areas such as automated negotiation, auctioning using agents, and the development of ontologies, which can be applied to all these classes of E-Commerce, are the focus of much research. This chapter covers the field of agents in E-Commerce broadly; however, since our application falls in the business-to-business category, we focus mainly on that area of E-Commerce.

5.2. Non-Agent-Based Electronic Commerce

5.2.1 Traditional Electronic Commerce

Traditional E-Commerce, that is E-Commerce before the Internet, has been in existence since the 1970's in the form of electronic fund transfers (EFT) and electronic data interchange (EDI). EFT enabled the direct payment of a sum of money from one bank to another, while EDI allowed for the exchange of standard business documents between businesses. EFT and EDI employed the public telecommunications network or value-added networks (VANs) for the transfer of data. These networks were closed systems where only those businesses that were part of the system could join the network. Such systems by their nature were more suited to long-term relationships between the participating businesses. They were also more suited to large businesses that could afford the software and hardware required to participate (e.g. adding a partner to a VAN could have cost more than $50000 in 1998 [Bolin, 1998]).

EFT and EDI are still widely used today, although the Internet has changed the way they are used. Modern Internet-based EDI systems are characterized by their lower cost and the more open nature of the systems they participate in. Relatively short-term relationships using EDI are now possible between small and large businesses, since Internet technology is widely available and does not incur a significant initial investment. It should be noted though, that not all businesses have migrated from traditional EDI as it is still more secure than Internet-based EDI.

5.2.2 Internet-based Electronic Commerce

The growth of the Internet and the World Wide Web in particular has resulted in new forms of E-Commerce. Many more business models have now been successfully implemented using the Internet. Some of these business models are defined by [Turban, 2002] and a selection is listed below ('The company' refers to the business entity implementing the model):

- **Name your price:** The buyer declares a price that he is prepared to pay for a product or service. The company tries to match the declaration with a supplier who can provide the product or service at the said price. E.g. Priceline.com
- **Find the best price:** The buyer defines his needs and specifications, and the company finds the supplier who can meet those needs and specifications at the best price. E.g. Hotwire.com
- **Dynamic brokering:** The company Webcasts the buyer's needs and specifications to suppliers as a tender. The suppliers then bid on the tender. Supplier bids can be revised until a suitable bid is received. The company automates the invitation to tender and the bidding process, and only requires the initial input from the buyer. E.g. www.GetThere.com
- **Electronic tendering systems:** Large-scale buyers put out tenders and receive bids on-line. This has recently been implemented in South Africa by the Independent Electoral Commission on the website www.votaquotes.elections.org.za

- **On-line auctions:** These are similar to 'real' auctions where buyers bid against each other for products or services. Bidorbuy.co.za is a local website that implements on-line auctions.
- **Electronic marketplaces and exchanges:** The company provides a site where buyers and sellers can meet and transact business. Electronic marketplaces are not a product of the Internet since most stock markets operate electronically. The Internet has, however, made it possible for electronic marketplaces to be open to many more participants. E.g. ChemConnect.com
- **Supply chain improvement:** The supply chain is made more efficient in one or more areas through the process of automation. This may involve electronic ordering, payment and production monitoring. E.g. www.productbank.com.au

Most of the models are applicable across the various classes of E-Commerce; however, certain models apply more to one class than another – for example, supply chain improvement and electronic tendering are more suited to business-to-business commerce, while others such as on-line auctions serve the business-to-consumer and consumer-to-consumer classes better. The models also address different phases of the E-Commerce life cycle – the 'Find the best price' model focuses more on the finding phase of the life cycle whereas dynamic brokering is more suited to the negotiation phase.

5.2.3 Enterprise Resource Planning Systems

Enterprise resource planning (ERP) systems seek to tackle the problem of multiple, isolated information systems that address the different areas of operation in large businesses or enterprises. An ERP system is defined as an integrated system of operation applications encompassing contract and order management, distribution, financials, human resource management, logistics, production and sales forecasting [Moodley, 2001].

ERP systems have developed from the material requirements planning (MRP) and MRP II systems that first came into use in the 1960's and 1980's respectively. MRP systems are defined as legacy systems that enable businesses to predict, track, and manage all the constituent parts or complex manufactured goods [Laudon & Guercio, 2002]. They are production planning tools that generate things such as master production schedules for manufactured goods, as well as bills of materials that specify the number and type of parts required to manufacture goods. MRP II systems are essentially MRP systems with management planning functionality. The MRP and MRP II systems, however, are not integrated with the other information systems of a business and are geared primarily towards manufacturing. ERP systems, in contrast, are integrated systems that include all the features of MRP II, and can be applied to many types of business.

MRP, MRP II and ERP systems have traditionally not been concerned with E-Commerce as we have defined it – they have been more focused on improving the efficiency within the business and not with interactions with entities outside the business [Moodley, 2001] (this is the reason MRP systems are discussed here and not in the earlier section on traditional E-Commerce). The manufacturers of ERP systems have only recently (since 1999) opened their systems and provided the ability to act with outside parties via the World Wide Web, mainly through Internet-based electronic marketplaces. These electronic marketplaces are primarily designed for business-to-business E-Commerce with businesses that use similar ERP systems.

5.3 Agent-based Electronic Commerce

Agent-based E-Commerce uses the unique properties of agents to extend the models of Internet-based E-Commerce. As the number of commercial entities that are on-line increases, so the amount of information and potential trading partners available also increases. The primary use of agents in E-Commerce is to represent the actors, generally buyers and sellers, in a commercial environment. Agents help reduce the cognitive load these actors face when dealing with these large amounts of information and multiple trading partners [Griggs, 2000] [Nwana et al., 1998]. Agents are also used at the infrastructure or system level to implement business models. These models may be implemented entirely as agent-based systems or they may employ agents only for certain tasks.

5.3.1 Classification Schemes for Agents in Electronic Commerce

The authors that have written overviews of agent-based E-Commerce have taken different approaches to categorizing the field. Nwana et al., [1998], Maes et al., [1999] and Turban et al., [2002] classify agents according to the different phases of E-Commerce they take part in. Ibrahim et al., [2001] believe that agents in E-Commerce can be categorized into one of three broad areas: agents as delegates, agents in markets and agents for business models. Papazoglou, [2001] provides a typology based on the roles they play in E-Commerce systems. We describe [Ibrahim et al., 2001] briefly since it is a broad classification scheme, and have a look at [Turban et al., 2002] and [Papazoglou, 2001] in more detail. Agent-based auction systems are dealt with separately since they are not explicitly described in the classification schemes presented here.

5.3.2 The Delegate, Market, Business Model Scheme

Ibrahim et al., [2001] presents a classification scheme we call the Delegate, Market, Business Model Scheme. It defines the actors, or human participants in an E-Commerce environment as buyers, sellers or intermediaries. Agents can then be used as delegates for the actors in one of two ways – the first involves the agent searching for the best deal for the actor, and the second involves the agent buying or selling on behalf of the actor. In both cases the agent may employ AI to help the actor make decisions. In the first case the agent is also implementing the 'Find the best price' business model.

Electronic markets employ agents as part of their infrastructure primarily for cooperation and coordination. Cooperation here refers to the use of agents for helping actors allocate sources, skills and products in the marketplace. Agents also help coordinate the trading activities within the market. They enforce the trading rules of the market that govern the behaviour of actors and their agents.

5.3.3 The Purchasing Life Cycle Scheme

Nwana et al., [1998], Maes et al., [1999] and Turban et al., [2002] base their classification on what we call the purchasing life cycle. The purchasing life cycle is derived from consumer buying behaviour models where six distinct phases can be abstracted [Terpsidis et al., 1997]. The phases are given different names by the different authors, but they are essentially analogous. They are (in order): need identification, product brokering, merchant brokering, negotiation, purchase and delivery, and product/service evaluation. An explanation of each phase is given, as well as an example of how agents are being used therein:

- **Need identification:** The buyer becomes, or is made aware of, the need for a certain service or product. Agents can be used here on the buyer's and seller's side. On the buyer's side, an agent can selectively obtain product or service information based on a profile of the buyer. On the seller's side, agents can also be used to profile buyers, and then preemptively send them information to make them aware of products or services they may find useful. An example of a seller side agent for need identification is Amazon.com's BookMatcher agent. This agent notifies users that make use of BookMatcher when a book that may wish to purchase arrives at Amazon.

- **Product brokering:** After the buyer has determined that a product or service is needed, the buyer enters the product brokering phase where he determines the particular product or service required. This involves matching products to needs, and evaluating different or competing products over certain criteria. The agents used in this phase called recommenders, use various methods such as analyzing using user data or recommendations to predict what products or services a user will prefer. Examples of such agents are those by GroupLens – a company that uses collaborative filtering for recommending movies [Good et al., 1999].
- **Merchant brokering:** Once a specific product or service has been chosen, the buyer needs to choose which merchant or supplier to buy it from. The buyer will have to evaluate merchants over multiple criteria. Merchant brokering using agents is achieved in the business-to-consumer domain with comparative shopping agents, also known as ShopBots. Excite's Jango ShopBot obtains product specifications from a buyer and then searches the Internet for matching products [Maes et al., 1999]. In the business-to-business domain, where business models such as electronic exchanges are more appropriate, agents like AgentWare's Syndicator can be used (it enables business-to-business electronic exchanges) [Turban et al., 2002].
- **Negotiation:** Negotiation occurs between the buyer and the merchant selected in the merchant brokering phase. It occurs mostly in the business-to-business domain since most business-to-consumer commerce involves fixed-price selling. It can involve any number of criteria, but is primarily concerned with price. Negotiation is one of the most heavily researched areas of agent-based E-Commerce and there are many examples of negotiation agents and systems. The Kasbah agent marketplace [Chavez & Maes, 1996] is frequently cited. It uses agents for negotiation between buyers and sellers in the consumer-to-consumer domain. Kasbah agents, however, only negotiate over price, and this limitation gave rise to the Tete-a-Tete system [Maes et al., 1999]. Kasbah and Tete-a-Tete both feature agents that only perform bilateral or two-party negotiations. Jennings et al., [1996] have developed the ADEPT agent system that allows for multi-party multi-issue negotiating agents. The ADEPT system was used by British Telecommunications in the business-to-business domain to automatically generate quotations for the design of networks [Faratin et al., 1998].

- **Purchase and delivery:** The purchase and delivery of a product or service involves receiving the product, or the completion of the service, and the subsequent payment. This phase does not necessarily begin after the negotiation has finished since it may be possible to continue negotiating while a service is being performed. An example of agents being used for purchasing is the SafeCheck system that prevents non-allowable cheques from being issued by performing authorization [Lee & Yoon, 2000]. Kasbah is capable of delivering electronic products such as software to the consumer.
- **Product/Service Evaluation:** The evaluation of a product or service may occur after a sale or service delivery, or it may be an ongoing process that occurs during the lifetime of the product or service. Agents can be used here to help automate feedback from the buyer. Firepond.com's Answer and Advice agent are used to reply to e-mail questions from customers [Turban et al., 2002].

5.3.4 Papazoglou's Typology

Papazoglou, [2001] lists four basic types of E-Commerce agents in his typology. A graphical representation is given below in Figure 15, followed by a brief overview of each type of agent.

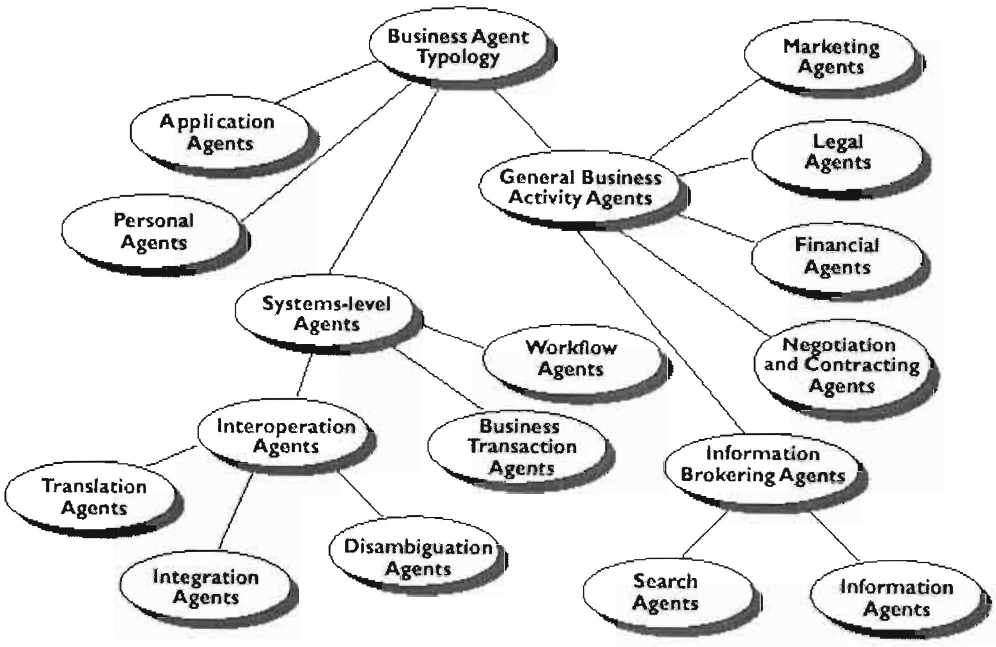


Figure 15 – Papazoglou's Typology [Papazoglou, 2001]

- **Application agents:** Application agents are agents that are specialized to a single area of expertise and provide access to the available information and knowledge sources in that domain. They may also collaborate with other agents to solve problems in that domain.
- **Personal agents:** Personal agents are analogous to interface agents from Nwana's agent taxonomy. These agents aid users in the presentation, organization and management of information relevant to the user's role in the business.
- **General business activity agents:** These agents perform a large number of activities in support of the general running of the business. They may be involved in searching for information, negotiating or marketing and can be customized to suit a particular business. Within this group there are two important types of agents, namely information brokering agents, and negotiating agents. Information brokering or matchmaking agents provide facilities for the location of other agents and information sources available on the network. They employ and maintain distributed directory services that list product and business services. Negotiating agents, also termed contracting agents here, serve the same function as they do in the Purchasing Life Cycle Scheme. They may take part in multi-party multi-issue negotiations with the goal of forming a contract with the opposite party or parties.
- **System-level support agents:** These agents provide the agent-level infrastructure for an agent-based E-Commerce system. They in turn are built on distributed object infrastructures such as CORBA. There are four important types of system-level agents – planning and scheduling agents, interoperation agents, business transaction agents and security agents. Planning and scheduling agents generate plans to coordinate the successful completion of a task by other agents. This generally requires some measure of AI. Interoperation agents allow an agent system to seamlessly interact with legacy systems. Agent-based E-Commerce systems are often developed after existing, or legacy systems, have been in operation. Interoperation agents serve as 'wrappers' for these legacy systems. Business transaction agents simplify the processing, monitoring and control of business transactions by automating activities such as transaction-related workflow and the enforcement of electronic contracts. Security agents maintain the security that

is essential in E-Commerce environment where financial information is transmitted along a network.

5.3.5 Agent-based Auction Systems

Traditional on-line Internet auctions involve a user accessing the auction website, searching for a product and then placing a bid on that product. The user must revisit the website to ascertain what the leading bid is and to place another bid. Agent-based auction systems extend traditional on-line Internet auction systems by automating stages of the auction process as well as by adding AI into the systems.

The most common extension is the use of an agent to represent the buyer. Buyer agents can reside on the auction server, or any other host, and bid on the user's behalf after the user has gone off-line. Also, buyer agents may employ user-defined bidding strategies and learn the strategies of competing agents in order to buy at the optimum price. Agents representing buyers do not only perform the function of a bidder – the BiddingBot agent system allows a buyer to send agents to many auctions and gather information on the price of an item so as to determine the fair market price of that item [Ito et. al, 2000].

The sellers in an auction can also be represented by agents – the Nomad agent system allows a seller's agent to begin an auction at the auction server [Sandholm & Huai, 2000]. System-level support agents may also be used in an auction system for things such as user agent authentication and the enforcement of auction rules (e.g. communication between colluding buyer agents would not be desirable).

6 The Distributed Buying Problem

6.1 Introduction

The distributed buying problem exists in businesses that are distributed over a large geographic area. During the merchant brokering phase of the purchasing life cycle, such businesses may have the additional option of procuring materials from other branches. The negotiation phase must then not only take into consideration the suppliers' lowest price, but also the cost of transporting the required material from other branches. An optimal execution of these two phases will ensure that the risk of duplicating purchases within the business, and the amount of time taken to contact multiple suppliers and branches, is minimized. We can therefore define the *distributed buying problem* as the business-wide optimization of the merchant brokering and negotiation phases of the purchasing life cycle. The use of agents in these phases is described in the purchasing life cycle scheme in Section 5.3.3.

In this research we focus on the use of mobile software agents to improve the merchant brokering and negotiation phases with respect to man-hours required and cost. BHP Billiton's aluminium-smelting operation in southern Africa is used as an example of a distributed business. It consists of three plants: the Hillside and Bayside plants in Richards Bay, South Africa, and the Mozal plant in Mozambique. The plants do not all produce the same end-products; however, their production processes do have common material requirements that create a distributed buying problem for the business.

6.2 The Role of the Buyer

In businesses that are sufficiently large, the function of buying materials is the responsibility of purchasing and supply management professionals. They have many job titles, but, for the sake of brevity, we use the term buyers to describe them. Buyers interact internally with individuals or departments that request materials, and externally with the suppliers that provide them. It is the buyer's responsibility to ensure that the business procures materials that best meet its needs in terms of cost, quantity, quality, or some combination of these or other criteria. The type and structure of the business determines to what extent it is the buyer that decides these criteria, or the individual or department that requested the material.

6.2.1 Types of Buying and Materials

Buyers may be involved in either *spot buying* or *strategic buying*. Spot buying refers to buying materials at a market price that is determined by supply and demand in a dynamic manner [Turban, 2002]. Strategic buying on the other hand, involves buying materials at a price that has been arrived at through a negotiated, long-term contract [Turban, 2002]. Note that spot buying does not imply buying from suppliers that do not have contracts with the business. Large businesses often deal only with suppliers that have registered with them and bound themselves to terms and conditions specified by the business.

The materials that are bought can be categorized as *direct* or *indirect*. Direct materials are used directly in production. Their use is scheduled, they are usually not shelf items, and they are bought in large volumes using strategic buying [Turban, 2002]. Direct materials are usually bought from suppliers with whom the business has an established relationship [Essmeyer, 2000]. Indirect materials, also called MROs, are used for maintenance, repairs and operations, and all other goods that are not directly used in production [Laudon & Guercio, 2002][Turban, 2002]. They are more easily bought using spot buying.

6.2.2 Traditional Buying Methods

The traditional method for buyers to interact with suppliers is a manual, paper-based one that requires the use of telephones, fax machines, face-to-face conversations and systems. This method is also currently the predominant method [Laudon & Guercio, 2002]. It places severe time restraints on buyers due to the slow nature of paper-based systems. Turban [2002] states that indirect materials constitute eighty percent of the total number of items bought by businesses, yet only twenty percent of the total value of the items bought. This forces buyers to spend more time focusing their efforts on low-value indirect materials, rather than on direct materials.

6.2.3 Procurement Reengineering

Procurement reengineering attempts to change the traditional method of buying to make it more efficient. This efficiency is usually achieved through the application of information technologies [Turban, 2002] and automation, but may also be achieved by a change in a company's manual buying processes. Most of the attempts at procurement reengineering have concentrated on spot buying and indirect materials – indirect materials because they constitute the bulk of buying, and spot buying because strategic buying, with its reliance on a greater deal of human-human interaction, is harder to automate/computerize.

Turban [2002] lists a number of sub-goals that can be contribute to the overall goal of efficient buying. The following are relevant to this work because the system developed here is targeted at these areas:

- Increasing buyer productivity
- Authorizing those who place buy orders with buyers to buy directly from their desktops, thereby bypassing buyers.
- Improving the flow and management of information between buyers and suppliers.
- Reducing human error in the buying process.

Businesses that have attempted to apply information technologies to procurement reengineering have used many different types of systems and E-Commerce models. In general, information technologies can automate manual tasks and greatly reduce paperwork. This saves the buyer's time and limits the introduction of human error in the process. The flow of information between the buyer and supplier is also made faster, since electronic documents can be transferred faster and managed more easily than their paper equivalents (this is the basis of EDI and workflow systems).

6.3 The Buying Process at Billiton

The buying process and systems are similar at the three Southern African Billiton aluminium-smelting plants. Each plant has a Commercial Division that is responsible for the buying of materials. The Commercial Division has commercial specialists, who are trained and experienced in buying materials related to specific operational areas.

Commercial specialists receive requests for the purchase of materials in the form of electronic documents, routed via e-mail or the SAP ERP system that is used at the plants. Requests originate from personnel working in the specialist's operational area of expertise. If the purchase exceeds a certain value, supervisory approval is needed and the document is routed to the supervisor, who may be a divisional superintendent, divisional manager or the plant manager, depending on the value of the purchase. Once the purchase has been approved, the document is routed to the specialist who then completes the purchasing life cycle from either the product or merchant brokering phases, or the negotiation phase (the request itself is the result of the need identification phase having already been undertaken).

The stage at which the specialist enters the purchasing life cycle is determined by how specific the request for the material (product) is. For example, an engineer might request a particular material from a particular merchant, in which case the specialist need only negotiate with that merchant. Alternatively, the engineer might only request a certain type of product, leaving the specific choice up to the specialist.

6.4 Solving the Distributed Buying Problem with Agents

To solve the distributed buying problem, we develop an agent-based proof-of-concept system that attempts to reengineer the buying process at each plant. The system uses mobile agents to circumvent the commercial specialists, and conducts the merchant brokering and negotiation phases of the purchasing life cycle itself. It makes use of the existing network and computing infrastructure that supports the SAP ERP systems at each plant.

Mobile agents are chosen for the following reasons:

- They are platform independent and may execute on a variety of different plant and supplier systems.
- Their mobility, autonomy, and asynchronous execution (See Chapter 3.5) allow them to be dispatched to other plants and suppliers systems without a persistent link to the dispatching agent system, and without the need for constant feedback from the buyer.
- To prevent duplication of resources and materials, it is necessary to perform remote queries on the materials databases of the other plants – mobile agents address this aspect of the distributed buying problem by their ability to perform efficient remote queries on large databases (See Chapter 3.5).
- They are capable of carrying out complex negotiation protocols in their negotiations with supplier agents (See Chapter 5.3.3).

In the next chapter we provide more detail on the system, and explain its design.

7 Design of the Mobile Agent System

7.1 Overview

A multi-agent system (based on the C/A/S model for mobile agents) is designed for a single plant, with the design then being replicated at the other plants due to the analogous organizational structure and operating procedures of all three plants. The result is an enterprise-wide multi-agent system that serves each plant individually and the enterprise as a whole.

The design diagram that follows represents the 'plant view', or the view of the multi-agent system operating at a single plant. It shows the overall architecture of the system and is not a detailed schematic of the exact functioning of the system at the plant level.

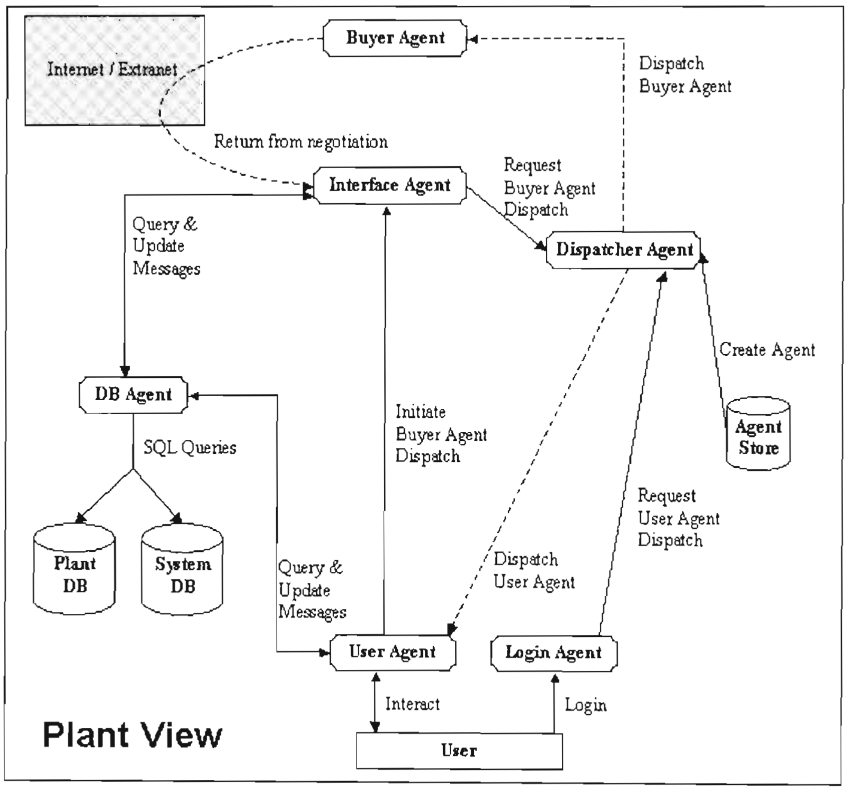


Figure 16 – Plant View (Directory & Finder Agents not shown for clarity)

The agent system at a plant is designed to be self-contained within the enterprise; that is, it may operate in isolation from the other plant agent systems in the enterprise. In order to function at its least effective level, it only needs to contact suppliers. The system consists of one mobile and four stationary *infrastructure agents* (i.e. agents that provide a service to the other agents). These agents are:

- *Database Agent* – provides access to the databases for *all* agents that require it.
- *Interface Agent* – provides an interface between the plant and outside agents.
- *Dispatcher Agent* – dispatches *Buyer* and *User Agents* to their destinations.
- *Directory Agent* – provides a directory service for locating the infrastructure agents.
- *Finder Agent* – locates the Directory, or any other specified agent.

The remaining agents are:

- *Login Agent* – a stationary agent that allows users to log in to the system.
- *User Agent* – a mobile agent that provides a GUI for the user to interact with the system
- *Buyer Agent* – a mobile agent that performs negotiations and places orders with suppliers.

The design can be viewed from an enterprise-wide perspective where the functioning of the system with other plants and suppliers is shown (Figure 17).

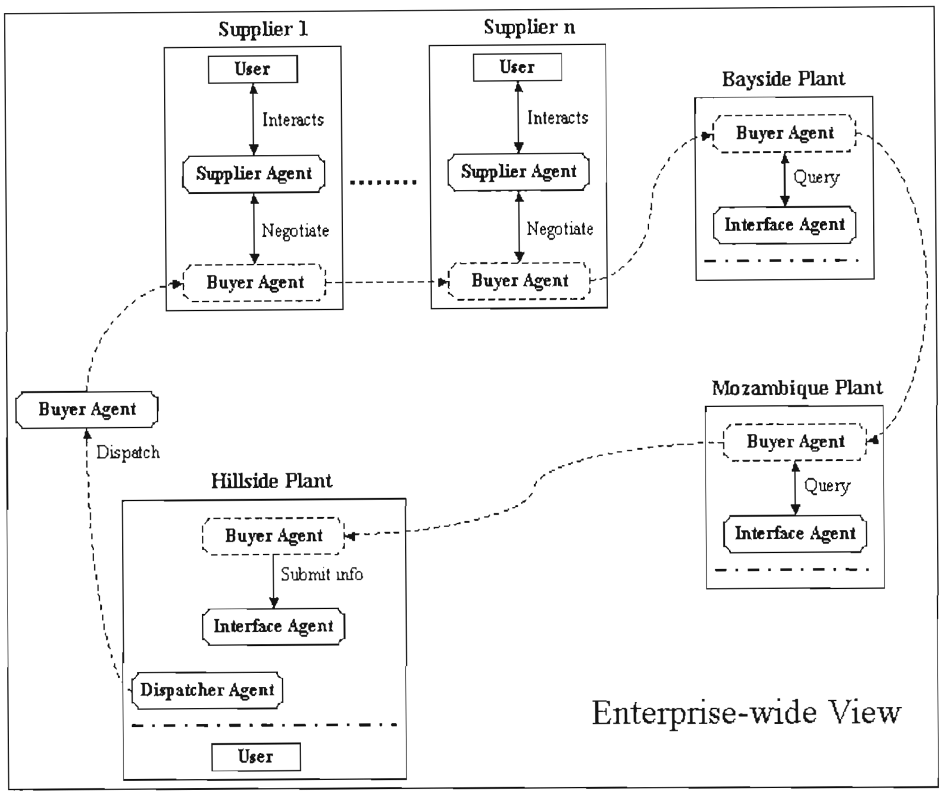


Figure 17 – The Enterprise-wide View

Figure 17 shows the dispatch of a Buyer Agent from the Hillside Plant and the path it traverses, visiting each supplier and the other plants in turn until it returns. The new elements introduced in this diagram are the Supplier Agents and the users that interact with them. Supplier Agents are those agents that the Buyer Agents communicates with during negotiations. They are the only agents that the design mandates on the supplier side. The supplier's agent system, apart from Supplier Agents, functions independently of the agent system presented here. All that is relevant to our design is that there is a Supplier Agent that is negotiated with, and that there is a user, an employee of the supplier, that provides feedback to this agent.

7.2 The Agents and their Roles

Login Agent

We begin our discussion of the agents with the Login Agent. The Login Agent receives information from a user, who is a buyer, or a member of a specific group of individuals authorised to initiate purchase requests. It has two functions in the system. The first is to provide a means for the user to authenticate to the system by providing details such as user name, password, and user group. The Login Agent performs the user authentication by checking the information against information contained in the database, via the Database Agent (this communication is not shown in the diagram).

If the authentication is successful, the User Agent is able to perform its second function of initiating the dispatch of the User Agent from the Dispatcher Agent, and providing the Dispatcher Agent with the parameters required to create and dispatch a User Agent. These parameters include the user's user-name, group and URL, as well as the URLs of the stationary infrastructure agents.

User Agent

The User Agent provides a graphical user interface (GUI) to the system for the user and serves as the user's main point of interaction with the system. It allows the user to:

- Perform queries on the database from a selection of queries that are available for his user group (the decision to provide a limited number of queries to each user group is based on the current practise at the three plants). Database queries are in fact queries to the Database Agent.
- Add queries to the list of possible queries.
- See the status of orders for materials used by his user group, which have been placed with suppliers and are awaiting delivery, and orders that are in the process of being negotiated using the agent system.
- Manipulate the negotiation process by accepting supplier quotes and placing orders for materials with the supplier, or initiating another round of negotiations.
- Request a quotation from a supplier by initiating the dispatch of a Buyer Agent to that supplier, or to all suppliers of that material (Note that in both cases the Buyer Agent is also dispatched to the other two plants to check stock levels there). In order to dispatch a Buyer Agent, the User Agent makes a request to the Interface Agent.
- Dispatch the User Agent to another URL. This is useful if the user is moving to another location in the plant. It would also allow the User Agent to reside on a mobile device that can support Java, GUIs and wireless TCP/IP.

Database Agent and databases

As part of the ERP system in place, a database management system (DBMS) contains the information describing users, materials, suppliers, quotes and orders. The agent system makes use of the DBMS by providing an agent wrapper – the Database Agent – for the relevant databases within the DBMS. The Database Agent facilitates all access to the databases by other agents.

All agents that need to retrieve, place new information into, or update the databases send messages to the Database Agent. It then effects the change or retrieves the data, and sends the result back. The messages are not SQL queries but are requests for a task to be performed, and the data necessary to perform it. The use of message passing between the 'parked' Database Agent and client agents fits the C/A/S model for mobile agents.

Wrapping the databases is preferable to providing each agent with the ability to interface directly with the databases, since a change to the database or interface to the database would necessitate a change to all the agents. In the case of wrapping, only the wrapper agent needs changing.

The system is designed such that the status of an order, or a negotiation process is, at any moment, recorded in the database. This is an important feature of the design as it:

- Maintains the state of orders and quotes in the system in the event of the system breaking down.
- Allows orders and quotes to be traced though the system to verify that it is operating properly.
- Provides records of the state of the database at different times so that potential disputes with suppliers over quotes or orders can be settled.

Interface Agent

The Interface Agent serves as the interface between the plant intranet, and the plant extranet and the Internet. All agents outside the plant's agent system that require interaction with the plant's agent system do so via the Interface Agent. A single point of interaction ensures that changes within the system are transparent to outside agents, and similarly, changes outside the system are transparent to agents inside the system. Only the Interface Agent needs to be changed in response to these changes. Furthermore, it is favourable to have one point of entry for agents entering the system, as it is easier to secure one point rather than many.

The Interface Agent handles User Agent requests for Buyer Agents to be dispatched. Before a Buyer Agent can be dispatched it must be recorded in the database that a request has been made for a Buyer Agent to be dispatched for a specific order, and, once the Buyer Agent has been successfully dispatched, this must also be recorded.

Having the Interface Agent handle requests for Buyer Agents also allows for different protocols relating to Buyer Agent dispatching. For example, it might be decided to collect all the orders or quotations from various user groups destined for a single supplier, and dispatch a single Buyer Agent with that information to the supplier. The Interface Agent also performs the task of resolving supplier names to URLs for dispatch requests received by User Agents. This ensures that if supplier URLs change, or a decision is made not to deal with a certain supplier, this change only needs to be made at one point.

All returning Buyer Agents report to the Interface Agent. The fact that a Buyer Agent related to a certain order or quotation has returned, along with the information it contains, is placed in the database by the Interface Agent (again, via the Database Agent).

The system is designed with the assumption that all suppliers have agent systems of their own with which to interact. It is possible to have the Interface Agent extend the system to handle the e-mailing of suppliers without agent systems. All that is required is for the Interface Agent to send a message to an E-mail Agent when it resolves the supplier name to a supplier without an agent system URL. The E-mail Agent can then dispatch an e-mail to the supplier, parse the reply and send this to the Interface Agent which will place it in the database.

Dispatcher Agent

The Dispatcher Agent is responsible for dispatching all Buyer and User Agents to their appropriate destinations. It acts as a single point from which other agents that request Buyer or User Agents dispatches can place their requests. Once a request has been made for an agent, the Dispatcher Agent creates the agent from the Agent Store with the parameters provided in the request, and dispatches it to its destination.

The reason for having a single point for dispatching is that updates to either Buyer or User Agents need only be made at one location. Furthermore, in the event that other agents are added to the system that need dispatching, they need only be stored once in the Agent Store.

Buyer Agent

The Buyer Agent performs automated negotiation and places orders with suppliers. It is created and dispatched by the Buyer Agent to the suppliers specified in the dispatch request. It visits each supplier in its list of URLs in turn, and then returns to the system where it communicates the result of its interaction with supplier agents to the Interface Agent.

The negotiation mechanism chosen involves sending the lowest quote from the previous request for quotes to each of the suppliers who bid. The user specifies the initial price for the first round of negotiations, and decides after each round whether to initiate another round. The user is left to decide how many rounds of negotiation are required in order to stop suppliers optimising their bidding strategies for a fixed number of rounds. This specific negotiation strategy was chosen for its simplicity; however, negotiation strategies can be made arbitrarily complex, with the only change being to the Buyer Agent, and possibly the Supplier Agent it interacts with.

Directory Agent

The Directory Agent plays a role similar to the Agent Management Service (AMS) in the FIPA standard – it provides agents with the network address and necessary contact information for the stationary infrastructure agents. These infrastructure agents, and any other agents that may need their location and contact information known to the system, sign in with the Directory Agent providing their name, URL and contact information. Thus, when other agents require information for a specific agent, they specify the name of the agent they are interested in, and the Directory Agent returns all the required information for that agent.

Since this design represents a proof-of-concept system there is a single Directory Agent. In an industrial setting one may have (without much additional complexity or changes to other parts of the system) multiple Directory Agents that update each other with the latest information, in much the same way as domain name servers and routers do.

Finder Agent

The Finder Agent's primary purpose is to find the Directory Agent for agents that require Directory Agent services. At some point in such an agent's initialisation, it will dispatch a Finder Agent to search for the Directory Agent. The Finder Agent traverses a sequence of URLs of possible Directory Agent locations it obtains from the dispatching agent, and returns as soon as it finds the Directory Agent. The list of URLs is held in a configuration file for the dispatching agent.

The Finder Agent has a secondary purpose of finding other agents when requested to by dispatching agents – all that is required is the name of the target agent and the list of URLs. The Finder Agent does not destroy itself once it returns with information but remains in case it is needed again at a later stage. This removes the overhead of creating a new Finder Agent during the dispatching agent's initialisation.

7.3 Discussion

The design shows that it is possible to reengineer the buying process using mobile software agents and hence solve the Distributed Buying Problem. As with any system design it is based on decisions that represent trade-offs between one aspect of design and another. An example is the significant assumption that suppliers have agent systems of their own with persistent connections to the Internet. This assumption was made to remove the complexity associated with handling suppliers that can be only be contacted by one medium or another (especially only by telephone or facsimile). It represents the general trend in procurement reengineering to move away from manual or paper-based methods of procurement to automated ones. The design is modular specifically so that it can be easily extended in order to deal with such complexities.

Having stationary infrastructure agents that provide functionality exclusively, such as a Dispatcher Agent being solely responsible for dispatches, means that a malfunction at this point disables that plant's system. The advantage, of course, is that changes to the dispatched agents, only need to be made at one point. The decision not to spread the functionality between agents, or have more than one of each type of infrastructure agent, is based on two reasons: the design is for a proof-of-concept system, and not an industrial system where reliability is paramount; a single point of failure is commonly found in many industrial client-server systems anyway, so our design is not necessarily less reliable.

8 Implementation of the Mobile Agent System

This chapter describes the implementation of a mobile agent system for the Hillside plant based on the design presented in the previous chapter.

8.1 Hardware and Software Used

The following software was used for development:

- The IBM Aglet Software Development Kit version 1.1.0
- The Sun Java Development Kit version 1.1.8
- The Sun Swing Classes version 1.1
- Borland's JBuilder 4.0 and its JavaBeans Component Library (JBCL) class for GUI development
- Microsoft SQL Server 2000
- Microsoft NT4 Server
- Red Hat Linux 7.1

IBM's Aglet SDK was chosen since it was freely available, had a large user base, and good developer support through its active mailing list. It is Java based, and at the time required version 1.1.x of the Sun JDK. Version 1.1.8 was selected since it was the last of the version 1.1.x releases.

Borland JBuilder 4.0 was used for the development of GUIs because its visual environment made the task much simpler. The Sun Swing classes for GUIs were used since they offered more functionality than the standard AWT classes. The Borland JBCL class was chosen to allow for the easier creation of more complex GUIs.

Microsoft's SQL Server 2000 was selected because it is both an enterprise-strength DBMS and free to use (due to the department's Microsoft Developer Network subscription). Moreover, it was easier to use and required less powerful hardware than the other freely available enterprise-strength databases tested.

The hardware used in the development consisted of two computers, connected by a TCP/IP network.

To prove that the concept of using mobile agents was indeed valid, the issue of heterogeneous operating environments was tackled. There are potentially many different operating systems in place within the enterprise and at suppliers, with the Microsoft Windows family of operating systems and Unix-type environments making up a large percentage of these. It was decided to use the Windows NT4 Server on one computer and Linux on the other to determine how easily mobile agents could migrate from one environment to another, and if there were any problems associated with this.

8.2 Databases and the System Dataflow

8.2.1 The Database Tables

The databases discussed here simulate the database services that would be made available to an agent system by the ERP system at a plant. Two databases are used in the implementation: the plant database and the user database. The plant database contains information describing a plant, the materials at that plant, orders and quotes for the plant, as well as the plant's supplier details. The user database contains information that is used to map available database queries to user groups. The most important tables in the plant database are the ORDERS and QUOTES tables. These tables are used to keep track of the state of orders and quotes in the system.

Table 3 shows the design of the ORDERS table (the transport cost is built into the total cost and it is therefore not included as a separate field). The DispatchCondition field records whether the Buyer Agent has been dispatched or has returned. The round of negotiation for which the Buyer Agent was dispatched or has returned from, is also recorded in this field. We are able to store these two facts by using a three-digit integer for the dispatch condition and setting the first digit to '1' when the Buyer Agent has been dispatched, and to '2' when it returns. The second two digits record the current round of negotiations and the number represented by the digits is incremented each time a new round of negotiations begins. The use of two digits obviously limits the number of rounds of negotiations to ninety-nine, however this is adequate for our purposes. Once a supplier quote has been accepted and the plant is awaiting delivery a single digit is used and set to '-1'.

The QUOTES table has essentially the same design as the ORDERS table. The QuoteDateTime field replaces the OrderDateTime field – it records the date and time the Buyer Agent received each quote. The Salesperson and Remark fields have also been added to record the name of the salesperson at the supplier that offered the quote, and any remarks he has regarding the quote. The QUOTES table only stores information regarding the last round of negotiations for an order.

Field Name	Purpose
OrderNumber	Used to store the order number of an order. The order number is a unique identifier for each order and therefore this is the primary key for the table.
MaterialCode	The code identifying the material ordered. Material codes are unique identifiers for materials in the MATERIAL table.
Quantity	The quantity of the material ordered.
Price	The price *per unit* of a material.
SupplierCode	The code of the supplier of the ordered material. Supplier codes are referenced from the SUPPLIER table.
Status	Shows whether an order has been confirmed and is awaiting delivery or whether it is still under negotiation.
MRPGroup	The user group that order is for.
DispatchCondition	Shows whether the buying agent dispatched for the order has returned or not, and what the current round of negotiation for the order is.
OrderDateTime	The date and time the order was placed by the user.
Controller	The user who placed the order.
TotalCost	The total cost of the order.

Table 3 – The ORDERS table in the user database

Reference is made in Table 3 to the SUPPLIER and MATERIAL tables in the plant database that describe all relevant information about the plant's suppliers and materials respectively. The MATERIAL table was populated using sample data from reports generated by the SAP ERP system at the Hillside plant.

The following figure shows the tables in the plant database.

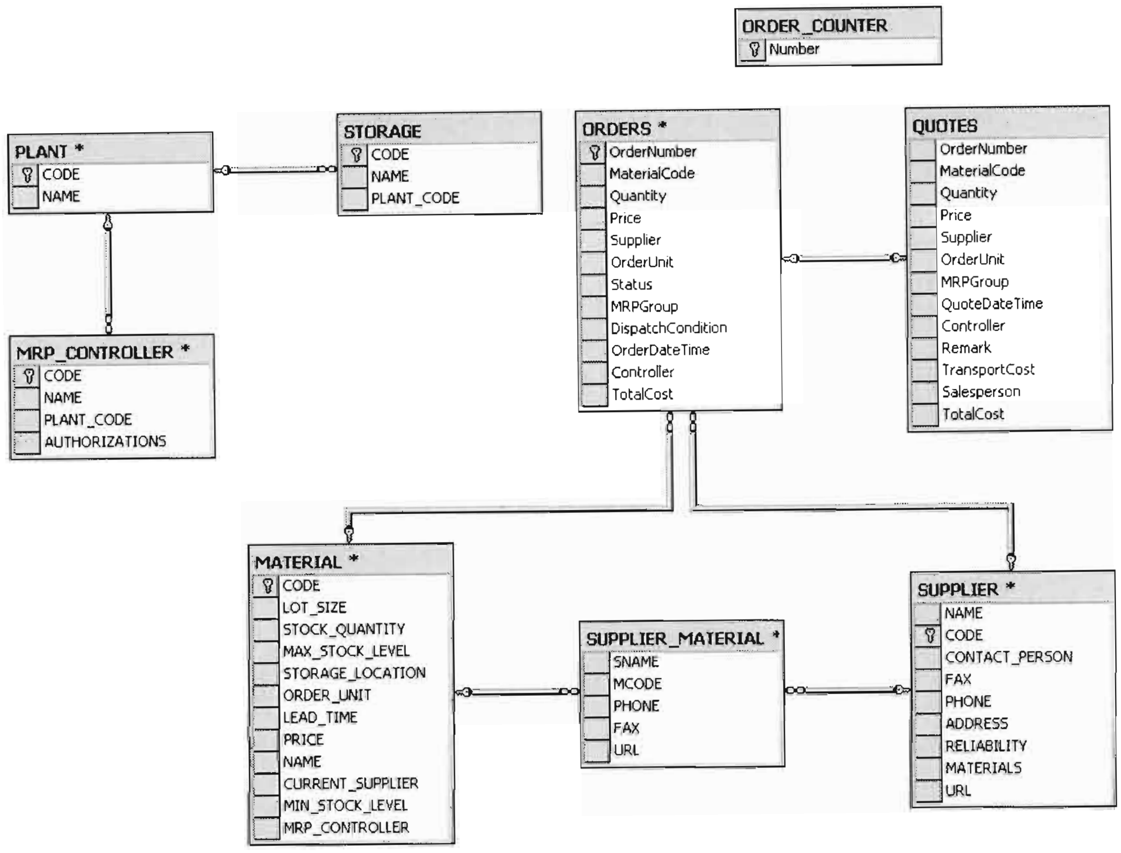


Figure 18 – The Database Schema for the User Database

8.2.2 System Dataflow

System dataflow refers to the flow of data through the system, and how this data is changed as it moves through the system. In our system the data of interest is the order and quote information. The structure of an order and quote does not change as it moves through the system (though the information describing the order or quote may change) therefore we can rely on the description of an order and quote as represented in the database.

Figure 19 shows the creation of a new order by a user using the User Agent and the source of each data element that makes up an order.

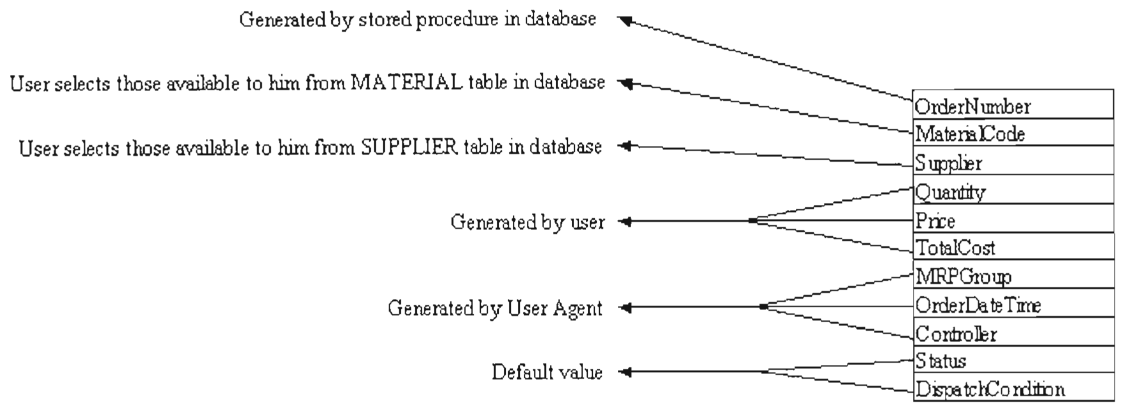

Generated by stored procedure in database
User selects those available to him from MATERIAL table in database
User selects those available to him from SUPPLIER table in database
Generated by user
Generated by User Agent
Default value
OrderNumber
MaterialCode
Supplier
Quantity
Price
TotalCost
MRPGroup
OrderDateTime
Controller
Status
DispatchCondition

Figure 19 – Order Creation

The order number for each new order is generated using a stored procedure in the database. The procedure merely increments the current order number (stored in the ORDER_COUNTER table in the plant database) by one. The materials and their suppliers are chosen by the user depending on what his user group is allowed access to in the database. The values themselves are obtained from the MATERIAL and SUPPLIER tables respectively. The quantity, price per item, and total cost of an order are determined by the user. The total cost may simply be the price per item multiplied by the quantity, or it may be some lower value if the user includes a discount. The date of an order (OrderDateTime), identity of the user (Controller) and user group (MRP Group) are provided automatically by the User Agent, since these are initialisation parameters for the User Agent. The order status and dispatch condition have default values in the database at the time of creation.

8.2.3 The Order and Quote Life Cycle

The lifecycle of an order and quote is illustrated below in Figure 20 and explained thereafter. It is used to show the system dataflow.

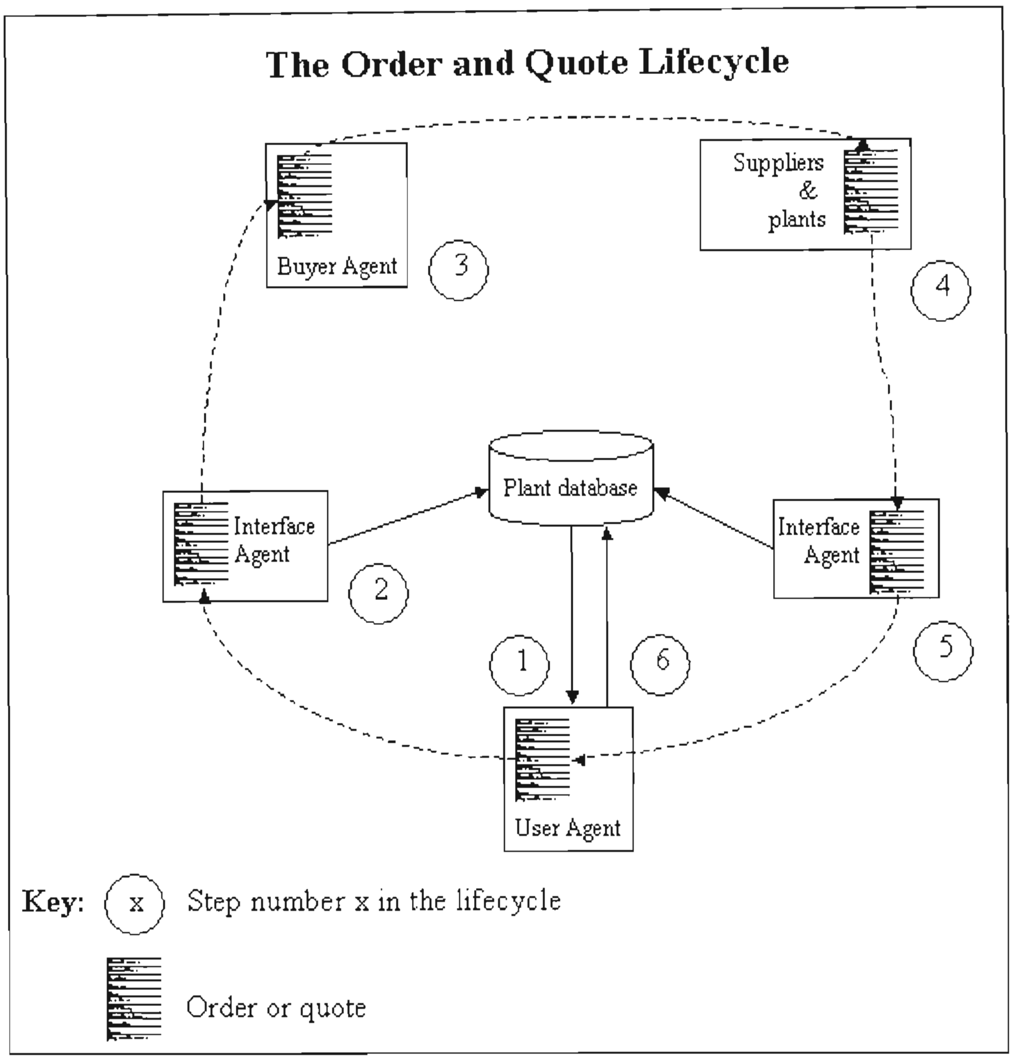


Figure 20 – The Order and Quote Lifecycle

1) The creation of an order. A new order may be created by the user as described in Figure 19, or an order may be generated from quotes stored in the database. The only difference to the creation process in the case of the latter is that all the data for the order is drawn from the QUOTES table rather than from the user and User Agent.

2) The Interface Agent changes the dispatch condition associated with the order to reflect that the Buyer Agent has been dispatched with the order information.

3) The order now takes the form of a quote in the Buyer Agent and is the basis of the request for quote (RFQ) the Buyer Agent will submit to the suppliers.

4) Quotes are obtained from suppliers and plants in response to the RFQ.

5) The dispatch condition of the order is again changed by the Interface Agent to indicate that the Buyer Agent has returned. The quotes associated with order are also stored in the database.

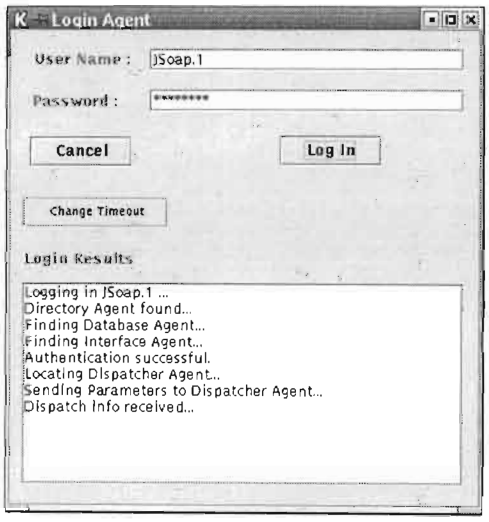
6) The acceptance of a quote. A change is made to the order status to reflect that the quote has been accepted and an order has been placed with the supplier. Once the material has been delivered, the record is removed from the ORDERS table.

8.3 Implementation of the Agents

Much of the complexity in developing the multi-agent system lay in the coordination of the different agents, all running their own threads of execution. Message passing was primarily used to coordinate the agents and for passing data between agents. Since Aglets do not support any ACL, an ad hoc message system was used.

Login Agent

The Login Agent achieves its functionality by providing a GUI (shown in Figure 21) for the user to log in with. The user enters his user name in the 'User Name' text field using the format `<database user name.user group number>`. The information is then parsed and, together with the password, is included in a message to the database agent requesting the authentication of the user. The Login agent is implemented in two classes: *loginAgent* and *userLoginFrame*.



```
Logging in JSoap.1 ...
Directory Agent found...
Finding Database Agent...
Finding Interface Agent...
Authentication successful.
Locating Dispatcher Agent...
Sending Parameters to Dispatcher Agent...
Dispatch Info received...
```

Figure 21 – Screenshot of the Login Agent GUI

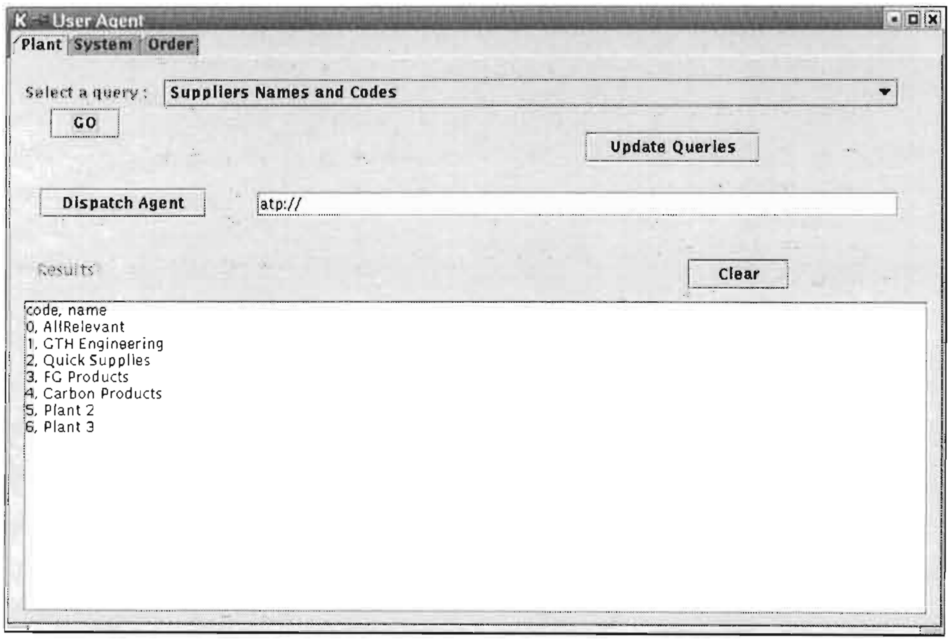
User Agent

The User Agent is implemented in the *userAgent* and *userAgentFrame* classes. It has a GUI consisting of three tabbed panes – ‘Plant’, ‘System’ and ‘Order’ – and a pop-up dialog box named ‘Order Status’. These are shown in Figures 22-25.

The ‘Plant’ pane allows the user to perform and update queries as well dispatch the agent to another location. The queries are available to the user in a drop-down list that contains only those queries that are meant for his user group.

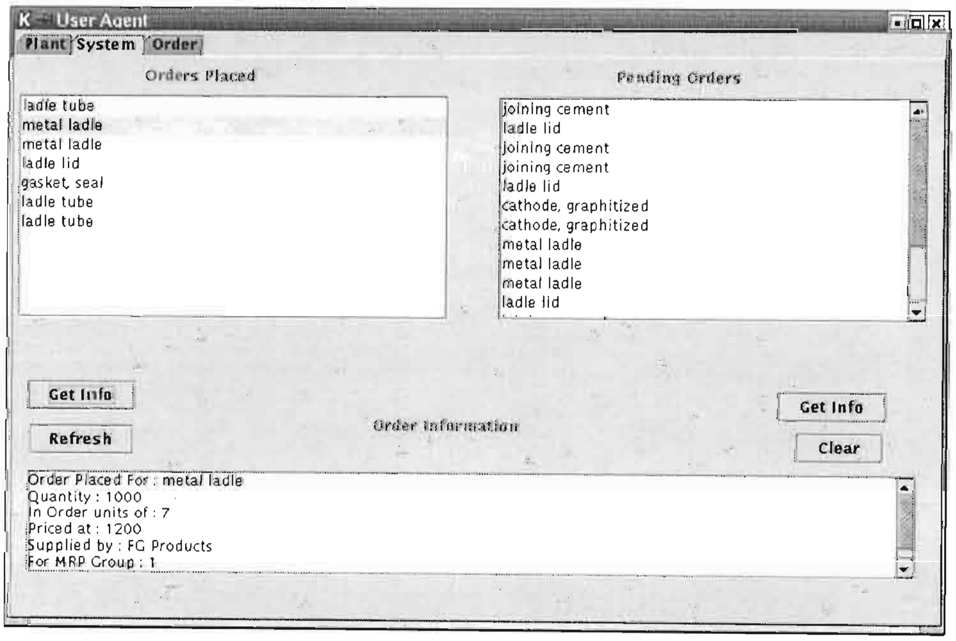
The ‘System’ pane contains two selection lists. The list entitled ‘Orders Placed’ lists orders that have been placed with suppliers and are awaiting delivery, while the ‘Pending Orders’ list shows orders that are in the process of being negotiated. The user can select an order in either list and have information about the order or pending order displayed in the text box at the bottom of the pane. The user may also double-click an order in the pending orders list to bring up an ‘Order Status’ dialog box that contains the information about each supplier’s quotation for the pending order in a selection list. From the dialog box the user can choose to send the order for another round of negotiations, or to place an order with the supplier selected from the selection list.

The ‘Order’ pane allows for the creation of orders and the dispatching of Buyer Agents for those orders. Two drop-down lists show all the materials available for the user’s user group, and the suppliers for those materials. The user can enter the price, quantity, and total cost in the text fields provided and submit this data by Buyer Agent to a single, or all suppliers, for negotiation.



```
code, name
0, AllRelevant
1, GTH Engineering
2, Quick Supplies
3, FG Products
4, Carbon Products
5, Plant 2
6, Plant 3
```

Figure 22 – Screenshot of the 'Plant ' Pane of the User Agent



Orders Placed:

```
ladle tube
metal ladle
metal ladle
ladle lid
gasket, seal
ladle tube
ladle tube
```

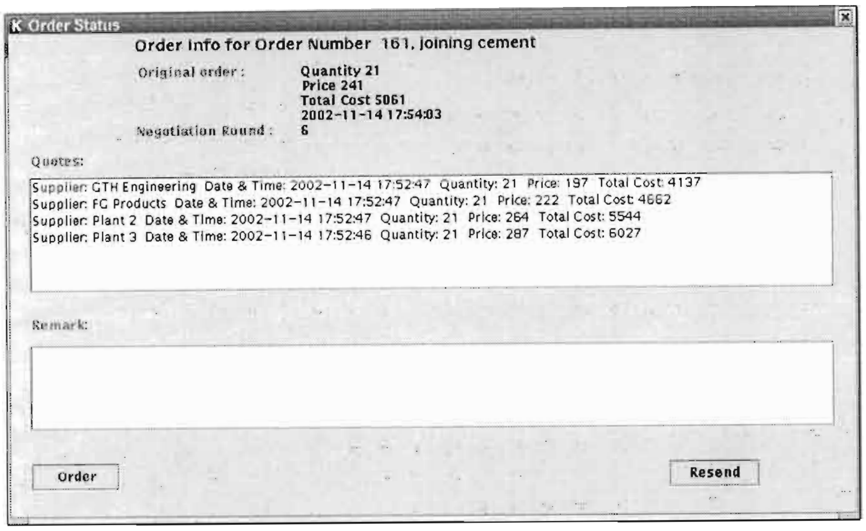
Pending Orders:

```
joining cement
ladle lid
joining cement
joining cement
ladle lid
cathode, graphitized
cathode, graphitized
metal ladle
metal ladle
metal ladle
ladle lid
```

Order Information:

```
Order Placed For : metal ladle
Quantity : 1000
In Order units of : 7
Priced at : 1200
Supplied by : FG Products
For MRP Group : 1
```

Figure 23 – Screenshot of the 'System ' pane of the User Agent



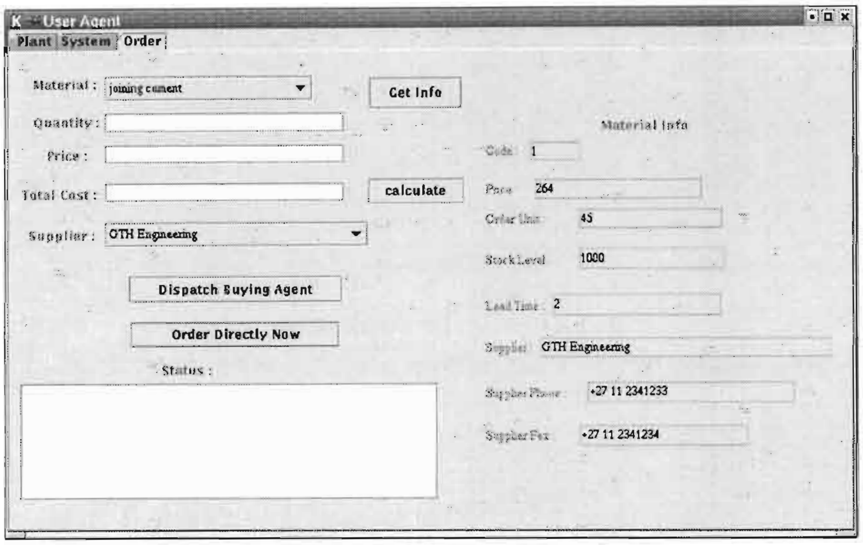
Order Info for Order Number 161, joining cement

Original order : Quantity 21
Price 241
Total Cost 5061
2002-11-14 17:54:03

Negotiation Round : 6

```
Supplier: GTH Engineering  Date & Time: 2002-11-14 17:52:47  Quantity: 21  Price: 197  Total Cost: 4137
Supplier: FG Products  Date & Time: 2002-11-14 17:52:47  Quantity: 21  Price: 222  Total Cost: 4662
Supplier: Plant 2  Date & Time: 2002-11-14 17:52:47  Quantity: 21  Price: 264  Total Cost: 5544
Supplier: Plant 3  Date & Time: 2002-11-14 17:52:46  Quantity: 21  Price: 287  Total Cost: 6027
```

Figure 24 – Screenshot of the 'Order Status' Pop-up Dialog Box



Material : joining cement

Quantity :

Price :

Total Cost :

Supplier : GTH Engineering

Code: 1

Price: 264

Order Unit: 45

Stock Level: 1000

Lead Time: 2

Supplier: GTH Engineering

Supplier Phone: +27 11 2341233

Supplier Fax: +27 11 2341234

Figure 25 – Screenshot of the 'Order' Pane of the User Agent

Database Agent

The Database Agent is run on the same host as the SQL Server DBMS. It communicates with the SQL Server via the Java Database Connectivity – Open Database Connectivity (JDBC-ODBC) bridge driver. The JDBC-ODBC driver is provided with the JDK for Java access to databases that support the ODBC standard. Once executing, it listens for incoming messages, which are placed in a queue and handled one at a time. Each message has a type, and each type is associated with a method in the Database Agent that is executed upon receiving the message.

As a typical example of message handling, the following code snippet shows a message of type 'authenticate' from the Login Agent and how it is dealt with (an explanation follows):

```
else if (msg.sameKind("authenticate")){
    Vector details = (Vector)msg.getArg();
    userName = (String)details.firstElement();
    pwd = (String)details.lastElement();
     if (authenticate(userName,pwd)){
      msg.sendReply("passed");
     }
     else {msg.sendReply("failed");}//else
  }
```

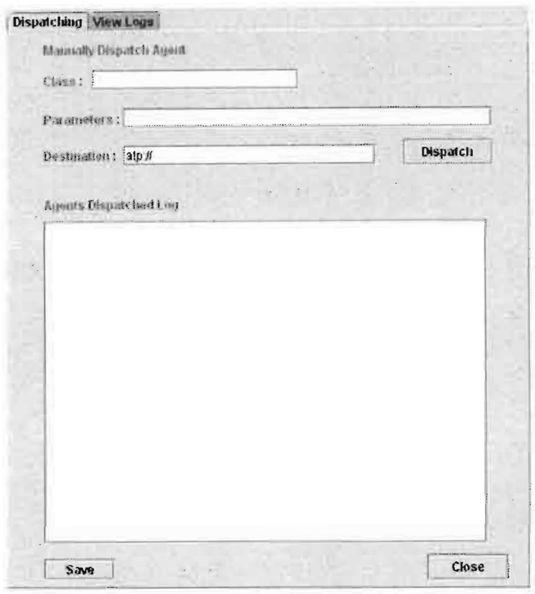
The message includes a single Java *Vector* object that contains the user's user name and password, which are obtained from the *Vector* object and passed as parameters to the boolean 'authenticate' method. If the 'authenticate' method evaluates to true, a reply to the message is sent indicating that the user has successfully authenticated, otherwise a reply is sent to indicate that the user failed the authentication process. The Database Agent is implemented in the *msSQLDBAgent* class.

Interface Agent

The Interface Agent is implemented in the *interfaceAgent* class. It is primarily a router of messages between the User and Dispatcher Agents, and the Buyer and Database Agents. Incoming messages are queued and handled one at a time. The messages are not simply forwarded when received but are dealt with by methods in the *interfaceAgent* class. Some of these methods message the Database Agent to store data in, and retrieve data from the database.

Dispatcher Agent

The Dispatcher Agent has a GUI (illustrated in Figure 26) that shows the agents that have been dispatched, as well as errors that have occurred during the creation or dispatching of agents. The time and date of these events, as well as the unique identifier of each agent, and the destination of the agent is also shown. The display is saved to a daily log file for record keeping and system debugging. The GUI allows for log files to be viewed in a separate pane. Agents can also be manually dispatched from the GUI. It is implemented in two classes: *dispatcherAgent* and *dispatcherAgentFrame*.



Destination : atp://

Figure 26 – Screenshot of the Dispatcher Agent GUI

Buyer Agent

The Buyer Agent is implemented in the *buyerAgent* class. Once created, it remains idle until it receives a message that marks the beginning of the dispatch protocol. The following diagram describes the dispatch protocol:

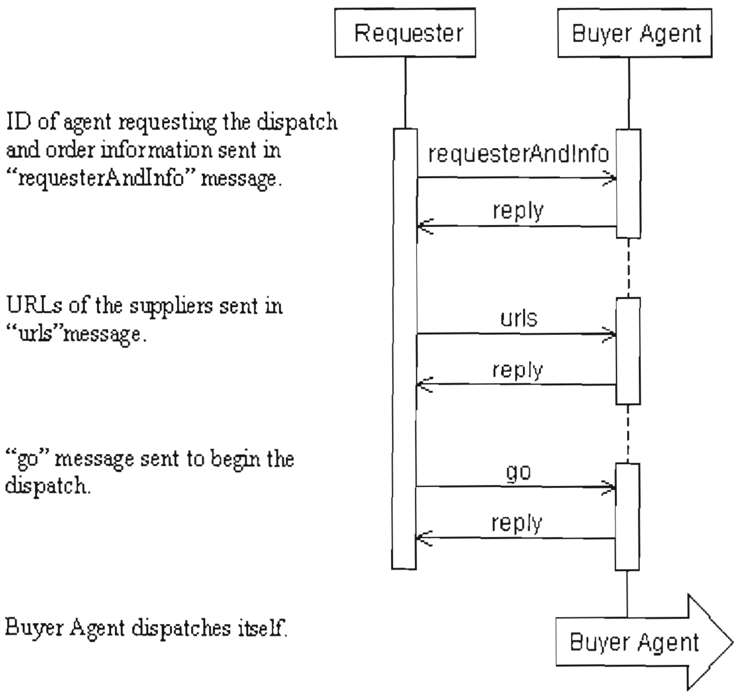

Requester
Buyer Agent
ID of agent requesting the dispatch and order information sent in "requesterAndInfo" message.
requesterAndInfo
reply
URLs of the suppliers sent in "urls"message.
urls
reply
"go" message sent to begin the dispatch.
go
reply
Buyer Agent dispatches itself.
Buyer Agent

Figure 27 – The Dispatch Protocol Used by the Buyer Agent

The Buyer Agent stores the order information in a *priceRequest* object that has attributes for each field in the QUOTES table of the database. When it arrives at a supplier, the *priceRequest* object is passed in a message to the Supplier Agent as a RFQ. The Supplier Agent then replies with the supplier's quote that is also in the form of a *priceRequest* object. The *priceRequest* objects are stored in a *Vector* object until the Buyer Agent returns to the plant, where the *Vector* object is transferred to the Interface Agent by message and stored in the QUOTES table of the database.

Directory Agent

The Directory Agent makes use of the Java *Hashtable* class (an implementation of a hash table) to store information about agents. To store their information with the Directory Agent, agents send it a "signIn" message that contains the agent's name and proxy object. The name and proxy object are stored in a *Hashtable* object as a key-value pair. Agents that require the URL of an agent, or any other details stored in a proxy object, send a "find" message to the Directory Agent. The "find" message contains the name of the agent, which is the key used to find the agent's proxy object in the hash table. The proxy is then returned in the reply. The Directory Agent is implemented in the *directoryAgent* class.

Finder Agent

The code for the mobility in the Buyer Agent is based on the Finder Agent. As such, the Finder Agent also uses the dispatch protocol shown in Figure 27 – the only difference being that agents send it a "requesterAndTarget" message that contains the name of the agent to be found (target agent) instead of a *priceRequest* object. It visits each URL in its list of URLs in turn until it finds the target agent, it then returns to its original location with the proxy of the target agent. The Finder Agent is implemented in the *finderAgent* class.

Supplier Agent

Supplier Agents are implemented to simulate suppliers and their agent systems. When a Buyer Agent sends the Supplier Agent a RFQ the Supplier Agent quotes a price that is lower, higher, or equal to the total cost specified in the RFQ. Each Supplier Agent is programmed to randomly select whether to quote lower, higher, or equal to the RFQ, with a certain probability and it quotes a random price that is within a certain percentage of the RFQ. The randomization of the quote generation was not done to accurately simulate how suppliers would quote, but to simulate a variety of different quotes from suppliers. The Supplier Agents are implemented in the *supplier?DBAgent* classes.

Plant Agent

Two Plant Agents are implemented in the *plant1DBAgent* and *plant2DBAgent* classes to simulate the other plants. They are exactly like Supplier Agents, with one Plant Agent programmed to quote higher than the other to reflect the higher transport cost involved in transporting materials from the Mozal plant.

8.3.1 Efficiency Issues

The User Agent relies on the Database Agent to provide it with all of its data, therefore there is frequent message passing between the two agents. The performance of the User Agent is thus governed by: the speed of the DBMS; the load on the Database Agent; its efficiency in handling requests; and the network latency between the two agents. We did not measure the performance in this work; however, it is worth noting that [Samaras et al., 1999] performed an analysis of the C/A/S model for database access using Aglets and reported an improvement of between thirty to forty percent over the traditional, client-server applet-based approach.

Aglets allows for multi-threaded parallel message handling that would improve performance in the Database Agent. The size of the system developed here did not warrant the additional complexity in dealing with multi-threaded database access; however, it would be preferable in larger systems. Further performance gains would also have been achieved through the use of stored procedures in the database, rather than interactive SQL methods in the Database Agent. This was not done since it would have bound the implementation of the agent system to a specific DBMS (stored procedures are written in procedural SQL, which can be DBMS-specific).

8.4 Configuring the Tahiti Servers

8.4.1 User Registration and Policy Files

As the Aglets agent system is implemented through the Tahiti server, the system requires a Tahiti server on every network host that agents execute on. Each Tahiti server needs to be correctly configured to allow agents originating from other Tahiti servers a context in which to execute. The same is true for agents representing users from other Tahiti servers. Tahiti manages access to itself and to the host resources through the use of a policy file and user database. The user database contains a list of users and their passwords, while the policy file specifies what access specific users and agents from specific code bases have to the host resources.

In our system, the registration of users was handled by using the ‘import user’ and ‘export user’ features in Tahiti. The ‘export user’ feature writes the user details to a ‘.id’ file, which must be copied to the file system of the Tahiti server where it is to be imported. This mechanism is feasible for a small number of users but poses a problem in a large system with many users. A possible solution may be to create a tool that generates ‘.id’ files from a master list of users, and to adapt the Tahiti server to import multiple users (the ASDK is open-source so adapting the Tahiti server is feasible).

The policy file of the Tahiti server is based on Java 1.2 policy file syntax. It consists of a series of grant entries, each of which contains a set of permissions. The syntax can be described by the following rule:

```
grant codeBase "URL", signedBy "signer_names"  {
   permission permission_class_name "target_name", "action",
    signedBy "signer_names";
    ....
   permission permission_class_name "target_name", "action",
    signedBy "signer_names";
};
```

The ‘signedBy’ field indicates support for digital signatures; however, we did not make use of these. An abbreviated form of the policy file used for the Tahiti server running on the Windows NT host is presented here. Some of the permissions that needed to be added to the default permissions are shown:

```
grant codeBase "atp://*:*/" {
  permission java.io.FilePermission "C:\\jdk1.1.8\\lib\\swing.properties", "read";
  permission java.net.SocketPermission "codebase:*", "connect";
  permission java.lang.RuntimePermission "loadLibrary.JdbcOdbc";
  permission java.lang.RuntimePermission "accessClassInPackage.com.borland.jbcl.*";
};
```

The grant entry shown here grants the permissions to agents from any code base using the Agent Transfer Protocol. The first permission listed is a file permission that allows the Java Swing properties file to be read. The second is a socket permission that allows a socket connection to the server to any agent from the code base in the grant entry. The last two permissions are runtime permissions that allow the dynamic link library called "JdbcOdbc.dll" to be used during runtime, and classes in the JBCL package to be accessed.

A single policy file for all Tahiti servers in the agent system could not be used due to the different file naming conventions used by the operating systems. As an example of a required policy file change, consider the first file permission shown in the abbreviated policy file – the same file permission for the Tahiti server running on the Linux host had to be written as:

```
permission java.io.FilePermission "/home/kamil/jdk118_v3/lib/swing.properties", "read";
```

It might seem that a possible solution to the problem is to use relative instead of absolute file naming; however, this did not work since Tahiti servers executing under Linux and Windows NT search different default directories for files with relative file names. Different policy files were therefore configured for each host. In a larger system a solution to the problem may be to have standard policy files for each of the different operating systems in use. The standard policy files could then be customised where necessary.

8.4.2 The Class Loader

Every time an agent is created in the Tahiti server (either by being created in, or migrating to the server), its class file, as well the class files for classes it uses, are dynamically loaded by the Tahiti server. To improve performance the class loader used in the Tahiti server maintains a cache of all previously loaded classes. When a class needs to be loaded, the class name is checked against those in the cache. If there is a match, the cached file is used. This presents a problem when agents or the classes they use are updated (as was frequently the case during the development of the system), since the new updated version is not loaded, but rather the old cached version.

In the case where we were required to create agents from within the Tahiti GUI, it was possible to use the 'Reload Class and Create' option to force the latest class to be loaded into the cache. The more difficult case to solve was that of agents not manually created by users – agents created by other agents, or agents arriving from remote destinations. To force the loading of the latest class files in these cases, it was necessary to use the 'Clear Class Cache Now' feature. This feature only clears the class cache when the server is shut down, so the server needed to be rebooted in anticipation of an updated class being required. The aglet API provides a method to flush the class cache; however, testing showed that this method did not work or was not supported by the Tahiti server.

9 Conclusion

In this work we examined the topic of software agents and mobile agents specifically. We began by surveying the literature regarding software agents, and, in doing so, addressed the fundamental question of what constitutes a software agent. We then narrowed our focus to mobile agents, which we introduced as a new paradigm for building distributed systems. The standards governing mobile agent systems were then presented to give a clear understanding of the theory behind their implementation. Some examples of these implementations were subsequently discussed.

A look at the use of software agents to solve problems in E-Commerce was the second thrust of this work. In particular we looked at using mobile agents to solve the Distributed Buying Problem; choosing BHP Billiton's Southern African aluminium-smelting operation as an example of a distributed enterprise. We reviewed traditional E-Commerce models and then described software agent-based E-Commerce. The Distributed Buying Problem was then defined; as well as the buying methods at the plants that make up Billiton's Southern African aluminium-smelting operations.

All of the work described thus far served as the theoretical basis for the mobile agent system that was then designed and implemented. The system's purpose was to solve the Distributed Buying Problem at the Billiton plants. The details of the design and implementation of the system were also discussed.

In general, we are of the opinion that software agents can make a significant impact on E-Commerce. Our literature survey suggests that most of the agents already in place in E-Commerce are prototype systems. This has much to do with the fact that agent technology is still in the developmental stage. We envisage that industrial strength agent technology will see an increase in the use of agents in E-Commerce.

We believe that the prototype developed in this work shows that it is possible to use mobile agents to solve the Distributed Buying Problem. The prototype; however, needs to be implemented on a larger scale, and tested under conditions that are more similar to those found in industry, before we can conclusively state that mobile agents are indeed an good way of solving the Distributed Buying Problem.

9.1 Future Work

To build on the results of this work, the following work can be carried out:

- A FIPA-compliant mobile agent system should be used to allow for the support of ACLs. This will not restrict the type of agent system being used at the supplier as the different agent systems can communicate using a standard ACL.
- Multi-threaded infrastructure agents should be designed and implemented to cater for heavier loads on the system.
- An applet-based interface to the agent-system could be developed. This will allow users to interact with the agent system in the more familiar web-browser environment.
- Various negotiation protocols can be experimented with to find an optimal one.
- Quantitative performance testing should be undertaken to determine if the agent system is fast enough for use in an enterprise environment.
- Security considerations should be dealt with to make sure that: order and quote data is secure while in transit with the agent; the agent system is secure and does not pose a threat to security of the enterprise computer systems.

References

(1998). Mobile Agent System Interoperability Facilities Specification, Object Management Group. **22-04-2002**.

Available at http://cgi.omg.org/docs/orbos/98-03-09.pdf

(2001). Cambridge Dictionaries Online, Cambridge University Press. **04-04-2002**.

Available at http://dictionary.cambridge.org

(2002). JADE Homepage, Telecom Italia Lab. **2002**.

Available online at http://sharon.cselt.it/projects/jade/

Bellifemine F, Poggi A, Rimassa G, (1999). JADE - A FIPA-compliant agent-framework. **2002**.

Available at http://sharon.cselt.it/projects/jade/papers.htm

Bellifemine F, Caire G, Trucco T, Rimassa G, (2002). JADE Programmers Guide, Telecom Italia Lab. **2002**.

Available online at http://sharon.cselt.it/projects/jade/

Bolin, S. (1998). "E-Commerce: A Market Analysis and Prognostication." <u>StandardView</u> **6**(3).

Available in the ACM Digital Library at http://www.acm.org

Bradshaw, J. (1997). An Introduction to Software Agents. <u>Software Agents</u>. J. Bradshaw, AAAI Press/ The MIT Press.

Available at http://agents.umbc.edu/introduction

Butte, T. (2002). "Technologies for the Development of Agent-based Distributed Applications." <u>ACM Crossroads</u>(Spring 2002).

Chavez, A. and P. Maes (1996). <u>Kasbah: An Agent Marketplace for Buying and Selling Goods</u>. First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London.

Available at http://citeseer.nj.nec.com/chavez96kasbah.html

Dennett, D. (1987). <u>The Intentional Stance</u>, The MIT Press.

Essmeyer, H. (2000). <u>E-Transaction Enablers Business Models, Trends and Issues</u>. European Business School Seminar, Oestrich-Winkel, Germany.

Available at http://citeseer.nj.nec.com/401398.html

Faratin P, Sierra C, Jennings N R, (1998). "Negotiation decision functions for autonomous agents." <u>International Journal of Robotics and Autonomous Systems</u> **24**(3-4).

Available at http://citeseer.nj.nec.com/faratin98negotiation.html

Franklin, S. and A. Graesser (1996). <u>Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents</u>. ECAI'96 Workshop on Agent Theories, Architectures, and Languages: Intelligent Agents III, Springer Verlag.

Available at http://www.msci.memphis.edu/~franklin/AgentProg.html

Genesereth, M. and S. Ketchpel (1994). "Software Agents." <u>Communications of the ACM</u>(July).

Available in the ACM Digital Library at http://www.acm.org

Good N, Schafer J B, Konstan J, Borchers A, Sarwar B, Herlocker J, Riedl J, (1999). Combining Collaborative Filtering with Personal Agents for Better Recommendations. Proceedings of the 1999 Conference of the American Association of Artifical Intelligence (AAAI-99).

Avalaible at http://www.cs.umn.edu/Research/GroupLens/publications.html

Griggs, K. (2000). An agent oriented business model for e-commerce based on the NYSE specialist system. ACM SIGCPR conference on Computer personnel research, Chicago, Illinois.

Available in the ACM Digital Library at http://www.acm.org

Harrison C G, Chess D M, Kershenbaum A, (1995). Mobile Agents: Are they a good idea? New York, IBM Thomas J. Watson Research Center, Distribution Services F-11 Stormytown, Post Office Box 218, Yorktown Heights, New York 10598.

Available at http://www.research.ibm.com/massive/mobag.ps

Ibrahim I. K, Schwinger W, Weippl E, Altman J, Winiwarter W, (2001). Agent Solutions for E-Business Transactions. 12th International Workshop on Database and Expert Systems Applications (DEXA 2001), Munich, Germany, IEEE Computer Society.

Available at http://citeseer.nj.nec.com/474092.html

Iglesias C A, Garijo M, Gonzalez J C, (1998). A Survey of Agent-Oriented Methodologies. Fifth International Conference on Agent Theories, Architectures and Languages, Paris, Springer-Verlag.

Available at http://citeseer.nj.nec.com/iglesias99survey.html

Ito T, Fukuta N, Shintani T, Sycara K, (2000). BiddingBot: A Multiagent Support System for Cooperative Bidding in Multiple Auctions. Fourth International Conference on MultiAgent Systems, Boston, Massachusetts.

Available at http://computer.org/Proceedings/icmas/0625/06250399abs.htm

Jennings N R, Norman T J, Faratin P, (1996). "ADEPT: an agent-based approach to business process management." ACM SIGMOD Record **27**(4).

Available in the ACM Digital Library at http://www.acm.org

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

Laudon, K. C. and C. Guercio (2002). E-Commerce: business, technology, society, Addison Wesley.

Lee, J. K. and H. S. Yoon (2000). "An Intelligent Agents-Based Virtually Defaultless Check System: The SafeCheck System." International Journal of Electronic Commerce **4**(3).

Maes (1995). "Artificial Life Meets Entertainment: Lifelike Autonomous Agents." Communications Of The ACM **38**(November 1995): 108.

Available in the ACM Digital Library at http://www.acm.org

Maes, P. and B. Schneiderman (1997). "Direct Manipulation vs. Interface Agents: a Debate." Interactions **4**(6).

Available in the ACM Digital Library at http://www.acm.org

Maes P, Guttman R H, Moukas A G, (1999). "Agents that Buy and Sell : Transforming Commerce as we Know It." Communications of the ACM(March 1999).

Available at http://ecommerce.media.mit.edu

Minar N, Gray M, Roup O, Krikorian R, Maes P, (1999). Hive: Distributed Agents for Networking Things. **2002**.

Available at http://www.hivecell.net/publications.html

Moodley, S. (2001). E-Business And Supply Chain Management In The Automotive Industry: Preliminary Findings From The Eastern Cape And Kwazulu-Natal Benchmarking Club Pilot Surveys. Durban, School of Development Studies, University of Natal Durban: 86-88.

Available at http://www.nu.ac.za/csds/publications.htm

Nwana, H. S. (1996). "Software Agents: An Overview." Knowledge Engineering Review **11**: 1-40.

Available at http://www.labs.bt.com/projects/agents/publish/papers/agentreview.htm

Nwana, H. and M. Wooldridge (1996). "Software Agent Technologies." BT Technology Journal **14**(4).

Available at http://citeseer.nj.nec.com/29693.html

Nwana H S, Rosenchein J, Sandholm T, Sierra C, Maes P, Guttmann R, (1998). Agent-Mediated Electronic Commerce: Issues, Challenges and some Viewpoints. International Conference on Autonomous Agents, Minneapolis MN USA.

Available in the ACM Digital Library at http://www.acm.org

Odell J, Kerr D, Laamanen H, Levine D, Mack G, Mattox D, McCabe F, McConnell S, Raatikaine K, Stout K, Thompson C, (2000). Agent Technology, Green Paper, Object Management Group Agent Platform Special Interest Group. **2002**.

Available at http://www.objs.com/agent/agents_Green_Paper_v100.doc

Papazoglou, M. P. (2001). "Agent-Oriented Technology in Support of E-business." Communications of the ACM **44**(4): 71-77.

Available in the ACM Digital Library at http://www.acm.org

Russell, S. and P. Norvig (1995). Intelligent Agents. Artificial Intelligence: A Modern Approach. Englewood Cliffs, NJ, Prentice-Hall.

Samaras G, Dikaiakos M, Spyrou C, Liverdos A, (1999). Mobile Agent Platforms for Web Databases: A Qualitative and Quantitative Assessment. Joint Symposium ASA/MA '99. First International Symposium on Agent Systems and Applications (ASA '99). Third International Symposium on Mobile Agents (MA '99), Palm Springs, California.

Available at http://citeseer.nj.nec.com/article/samaras99mobile.html

Sandholm, T. and Q. Huai (2000). "Nomad:Mobile Agent System for an Internet-Based Auction House." IEEE Internet Computing **4**(2).

http://computer.org/Internet/ic2000/w2080abs.htm

Shoham, Y. (1993). "Agent-oriented programming." Artificial Intelligence **60**: 51-92.

Stuurman, K. and H. Wijnands (2001). "Software Law. Intelligent Agents: A Curse Or A Blessing? A Survey Of The Legal Aspects Of The Application Of Intelligent Software Systems." Computer Law & Security Report **17**(2).

Suri, N. (2002) Personal communication. Institute for Human and Machine Cognition, University of West Florida, 40 S.Alcaniz St., Pensacola, FL 32501

Sycara K, Decker K, Pannu A, Williamson M, Zeng D, (1996). "Distributed Intelligent Agents." IEEE Expert(December): 36-46.

Available at http://www.cs.cmu.edu/softagents/publications.html

Terpsidis I S, Moukas A, Pergioudakis B, Doukidis G, Maes P, (1997). The potential of Electronic Commerce in re-engineering consumer-retailer relationships through Intelligent Agents. Advances in Information Technologies: The Business Challenge. J.-Y. R. e. a. (eds.), IOS Press.

Available at http://citeseer.nj.nec.com/terpsidis97potential.html

Turban E, Lee J, King D, Michael Chung H, (2000). Electronic Commerce: A Managerial Perspective. New Jersey, Prentice-Hall.

Turban E, King D, Lee J, Warkentin M, Michael Chung H, (2002). Electronic Commerce: A Managerial Perspective. New Jersey, Prentice-Hall.

Van de Velde, W. (1995). Cognitive Architectures - From Knowledge Level to Structural Coupling. The Biology and Technology of Intelligent Autonomous Agents. L. Steels. Berlin, Springer-Verlag: 197-221.

Available at http://citeseer.nj.nec.com/vandevelde95cognitive.html

Wayner, P. and A. Joch (1995). Agents of Change. Byte. **March:** 94-95.

Wooldridge, M. and N. Jennings (1995). "Intelligent Agents: Theory and Practice." Knowledge Engineering Review(July).

Available at http://citeseer.nj.nec.com/97055.html