Abstract

A descriptive overview of the use of the term “agent” is presented. Some existing definitions of “software agents” are analyzed and some distinctions between software agents and other types of software using non-anthropomorphic criteria are suggested. Mobile agents and speech-act agents are discussed as special cases of distributed computing. A test-case application implemented on two Java-based platforms for mobile agents is presented. Some applications of agent technology in electronic commerce are presented. Business logic conditions for the deployment of an open agent-based market for automated electronic commerce is discussed.

Referat

Perspektiv på agenter -
En översikt med tillämpningar inom elektronisk handel

En deskriptiv översikt av bruket av termen ”agent” presenteras. Några definitioner av termen ”programvaruagenter” analyseras och några icke antropomorfistiska kriterier för att särskilja programvaruagenter från annan programvara föreslås, liksom kriterier för att kunna särskilja olika typer av programvaruagenter. Mobila agenter och talaktsagenter diskuteras som särfall av distribuerade programsystem. En testapplikation implementerad på två Java-baserade plattformar för mobila agenter presenteras liksom några tillämpningar av agentteknik inom elektronisk handel. Affärsmässiga förutsättningar för spridandet av ett öppet agentbaserat system för automatiserad elektronisk handel diskuteras.

Preface

This document is a Master’s thesis in computer science with which I complete my studies at the Department of Numerical Analysis and Computing Science (Nada) at Stockholm University, Sweden. My Master’s project has been carried out at the Swedish Institute for Information Technology (SITI) as a part of the Tahuti project [SITI 1999]. My Nada supervisor has been Kjell Lindqvist. My supervisor at SITI has been Thomas Alexandre, project manager of the Tahuti project. For their patience, support and inspiration I thank them both. Sverker Janson and Joakim Eriksson at Intelligent Systems Laboratory at Swedish Institute of Computer Science (SICS) have provided me with valuable material on agents and on their work on the Agent-Based Market Space. Helene Wallgren at the Electronic Commerce Lab at SITI has supplied a section regarding legal aspects on agent technology for which i thank her dearly. I also would like to thank the rest of my new friends and colleagues at SITI and in particular the team at the Electronic Commerce Lab: Bernd Stadler, William Song, Mathias Bjarme, Stina Eklöf and Patrik Axelsson. Judith Chrystal has reviewed the manuscript and given valuable comments on my English.

Examiner is Professor Johan Håstad, Nada

Kista 31 May 1999

Jonas Edlund

Contents

Words followed by asterisk* are explained in the glossary.

1 Introduction

“My name is Agent, Software Agent"

In the 1990s there has been much talk about software agents. Future scenarios have been given where software agents help us with our daily lives in many different ways. They help us buy a new car or to make reservations for our holiday trip. They sort our mail and they help us to find what we are looking for on the WWW – or what we do not yet know that we are interested in. The expectations on software agents a few years ago were very high. It was thought that they should act as mediators or communicators between the naïve end-users and the sophisticated and incomprehensible software of today and tomorrow. And they should act as glueware between otherwise incompatible software systems. Many of these expectations have – as yet – not been fulfilled, but much promising work is underway.

In the world of software the use of the word agent became frequent in the AI heydays of the 1970s and 1980s. We got Artificial Intelligence, Thinking Machines and Intelligent and Autonomous Agents. The vision of software agents as our servants (or slaves) that we can create and destroy at will and that obediently fulfill our wishes was, and is, very compelling. It is also confusing. Computer programs do not think or act autonomously. They are executed on computers.

However disturbingly animistic* the terms may be, “Intelligence”, “Autonomous” and “Agent” are probably in computer science, or at least in the software industry, to stay. They are useful as abstraction mechanisms if you know that that is what they are and if you know their use and meaning.

1.1 Problem

- What are software agents?
- What do they do?
- How do they work?
- What is agent technology or agent-based programming?
- What kinds of applications are suitable for agents?

- What do software agents do in electronic commerce?
- What can they do in electronic commerce?

The original assignment for this thesis work was “Intelligent Agents in Electronic Commerce”. The first and obvious question was “What are software agents?” Investigations revealed the term “agent” as ambiguous. It has been applied to many different phenomena, many of which, when more closely examined, have little to do with each other.

“Agent” is a buzzword used in sales as well as in fundraising and often its use displays a strong tendency towards unnecessary anthropomorphisms. This leads to high expectations that can be exploited on a short basis but there are no gains to be made in the long run. A potent term like “agent” can be very confusing if not defined rigorously. Inexact concepts always result in bad science.

1.2 Aim

- Useful concepts, definitions and distinctions
- Categorization and taxonomy
- Understanding of the underlying technology
- Overview of existing applications in automated electronic commerce

1.3 Method

We have tried to sort out the terminology and look behind it to the underlying software technology to see what it is, how it works and what it has been and what it can be used for. We have concentrated on four topics:

- A concept analysis of “agent” and “agency” by a survey and critique of usage and definitions of the terms. We investigate how usage and meaning of the terms differ in different contexts.
- A closer study of mobile agents and speech-act agents as special cases of distributed computing,
- A brief study of impersonating agents,
- Some example applications in electronic commerce and entertainment.

1.4 Organization

Section 1
An introduction presenting problem, aim and method of the paper.

Section 2
An overview of software agents with focus on usage of the terms “agent” and “agency” and analyses of some proposed definitions. Some non-anthropomorphic criteria on agency are discussed.

Section 3
A technical overview of distributed computing. It can be skipped if the reader has previous knowledge on the subject.

Section 4
A discussion on mobile agents as a special case of distributed computing with special focus on Java-based platforms

Section 5
A description of a test case application and its implementation on two Java-based mobile agent platforms.

Section 6
A discussion on speech-act-based agents.

Section 7
An overview on applications in electronic commerce with focus on the business-to-consumer domain.

Section 8
Some examples of impersonating software agent applications in entertainment are presented. The distinction between matter and spirit is discussed in the perspective of the artificial life metaphor.

Section 9
Some concluding remarks.

Glossary
Words followed by asterisk* are explained in the glossary.

2 “Agent” and “Agency” – usage and definitions

The (agent) metaphor has become so pervasive that we’re waiting for some enterprising company to advertise its computer switches as empowerment agents.

[Wayner, P. & Joch, A., "Agents of Change", Byte, March 94-95, 1995, p. 95.]

The concepts “agent” and “agency”, the property of being an agent, are used in very different ways and in different context by different authors. In this section we will look at usage of these two terms and different complementing attributes that often come along with them, as “intelligent”, “autonomous”, “mobile” etc. We will analyze how this terminology can or cannot be used to distinguish between different kinds of software. Are there any criteria that make software agents distinct from other software? We will look at some suggested definitions and suggest some distinctions of our own.

2.1 A Dictionary’s View of Agents

In the Merriam-Webster [1999] dictionary, we find the following definitions of the word “agent”:

1 One that acts or exerts power

2a Something that produces or is capable of producing an effect: an active or efficient cause

2b A chemically, physically, or biologically active principle

3 A means or instrument by which a guiding intelligence achieves a result

4 One who is authorized to act for or in the place of another: as

- a representative, emissary, or official of a government <crown agent> <federal agent>
- one engaged in undercover activities (as espionage) : SPY <secret agent>
- a business representative (as of an athlete or entertainer) <a theatrical agent>

[Merriam-Webster 1999]

Later in this paper we refer to these as MW1, MW2a etc.

Someone who acts, something that produces an effect should thus qualify as an agent. According to this definition, “agent” is a very wide concept. MW2a cover many inanimate objects, especially in unexpected situations like accidents: “The needle made the balloon burst”. MW3 covers most artifacts: “The parachute saved my life”.

If we concentrate on MW1 and MW4 we see that an “agent” is someone or something that is potent, active, autonomous, hopefully at your service and under your control, but also possibly unpredictable and potentially harmful. Does this also apply to software agents?

2.2 A Descriptive View of Agents

“Agent” is a precious term and many different parties want to reserve it for their own particular definition. We take a descriptive stance and investigate the meaning of the terms by looking at how they are used.

2.2.1 Non-Software Agents

If we start out from a descriptive, unbiased position and look at the use of “agent” and “agency” outside the world of software, we find examples like

- Agent/Impresario – agent of artists, helps the artist in business relations, e.g. gives help in examining and closing contracts, a business representative,
- Travel Agent – helps with travel arrangements,
- Sewing Machine Agent – sells you a sewing machine,
- Booking Agent – can book an artist for a theater
- Free Agent – is free to act in whichever way he or she likes,
- Active Chemical Ingredient – the ingredient that makes it happen,
- Leavening agent (q.v.) used in making baked goods – makes your dough rise.
- James Bond – secret agent, needs no further introduction.

2.2.2 Software Agents

In the world of computer programs we see “agent” used in many different ways. Here are some examples with preliminary definitions:

- **Software agent:** a program or a piece of code that is also an agent,
- **Intelligent agent:** program that exhibits intelligent behavior. Intelligence remains to be defined,
- **Autonomous agent:** Autonomy remains to be defined,
- **Adaptive autonomous agents:** Adapts to its environment,
- **Mobile agent:** Migrating code, code that moves from one machine to another as a part of its execution,
- **Impersonating agents:** software that projects a believable persona on to the user/audience,
- **User-interface agents**: software that helps a user to be more effective. If a word-processor was equipped with a such an agent it could suggest to the user more effective ways to accomplish something that the user often does, e.g. by writing a macro definition to change between two different fonts,

- **Typed-message-agents:** agents that communicate with typed messages
- **Speech-act agents:** a type of typed-message agents modeled on modeled on natural speech
- **Coordinated agents** or
- **Multi Agent Systems:** A group of software agents cooperate to solve a (common) problem.
- **Assistant agents:** The function of an assistant agent is to make a system friendlier to the end-user. It may help with mail sorting, information filtering, scheduling, finding interesting webpages etc.

These categories are not disjunct, most “agents” fall into several of them.

2.3 Linguistics: “Agency” as Conscious Action

In linguistics the “semantic agent” in a sentence refers to the initiator of the action described by the verb. In the sentence “Joe kicked the ball”, “Joe” is the semantic agent. Agency refers to *animate* action, i.e. the one that acts is or *can be thought to be* (our italics) conscious of or able to influence the action, as opposed to *inanimate* action which is referred to as “cause”. [Jörgensen and Svensson 1986, p88].

Add to this the Merriam-Webster definition and we see that “agent” is an enormously wide concept: one that acts, an instrument to accomplish a task. And as we shall see, almost anything can be an agent.

2.3.1 “Conscious”

Can software be thought of as being conscious? Maybe a naïve end-user can, in frustration over a perceivably hostile program. “This ------ program!!! If you do that again, I´ll kill you!” But for the most part we do not think of software as being conscious. We can *talk* about software as if it is conscious, as a matter of fact we do that often: “The program expects you to type your e-mail address”, “The program believes you want all mail from siti.se in the SITI folder”. If an entity like a computer program behaves as if it had certain beliefs or intentions one can say in an informal way that it has those beliefs and intentions, even if we understand that there is no one “inside” the program, knowing or doing anything. This is quite close to the “New Turing Test” of Hayes-Roth et al. [1999]. This way of thinking about software was elaborated and to some extent formalized by John McCarthy [1979]

> To ascribe certain beliefs, knowledge, free will, intentions, consciousness, abilities or wants to a machine or computer program is legitimate when such an ascription expresses the same information about the machine that it expresses about a person. It is useful when the ascription helps us understand the structure of the machine, its past or future behavior, or how to repair or improve it. *It is perhaps never logically required even for humans*, but expressing reasonably briefly what is actually known about the state of a machine in a particular situation may require ascribing mental qualities or qualities isomorphic to them.

(Our italics)[McCarthy 1979]

Can “conscious” be used to distinguish software? Is there conscious software and unconscious software? No, there is not. “Conscious” is not a property of any software. It belongs to the domain of Experience of Software or Thoughts about Software, and can thus be relevant in design and use of software.

2.3.2 “Acting”

Does software act? In the meaning “changing the state of its environment”, most software do. That is why we use it. But as with “conscious”, it is pointless to distinguish acting software as agents and non-acting software as non-agent if we look only on the implementation. “Acting” also belongs to Thoughts about Software and can be useful in the design and use phases.

2.3.3 Agency: A Metaphor

There is no consciously acting software. So however the terms are used, “agent” and “agency” are metaphors. Software agents may act *as if* they were “conscious of what they were doing”. In the following we will examine how these metaphors are used and whether or not they have a useful meaning in computer science.

2.4 Agency in Design, Implementation and Use of Software

The Norwegian researcher Christen Krogh of SINTEF Telecom and Informatics has made some very useful distinctions. He points out that the agent metaphor is used in very different ways in different phases of the software lifecycle. We have changed his original distinctions slightly.

- **In design:** to denote logical parts of a system that has some kind of autonomy,
- **In implementation:** to denote executable objects with agent characteristics,
- **In use of software:** The end-user interface is described with the agent metaphor:
- **In marketing:** The agent metaphor is used for selling the system,
- **In end-user training:** The end-user is encouraged to think of the system as an autonomous agent,
- **In experiencing or talking about the system:** The system is experienced or talked about as an autonomous agent.

[Krogh 1996].

The “IDs” of the Amalthaea info-mining system described in section 7.4 is an example of agents at the design level as are the agents of the trading systems in section 7.2. The speech-act agents in section 6 or mobile agents in section 5 are examples of agents at the implementation level as are Creatures of section 8.2.1. The rest of the impersonating agents described in section 8 fall in the category of interface-level agents.

The actual software is present only at the level of implementation, i.e. software consists of executable code. Modeling and design, however useful, belong to the domain *thoughts about software* as do the experiencing of software user interfaces.

This has relevance for the discussion on agency as well as for that on AI*. Much of the controversy over the question whether or not machines really can be autonomous agents or intelligent entities resolves if we make distinct in which phase we apply the concept, in design, in implementation or in use. Compare to how we look at ourselves. Do we see intelligence at the level of neurons?

2.5 Attributes of “Agency”

In this section we examine some of the attributes that are frequently used together with “agent” and “agency”. We start with a close examination of some of the distinctions of one of the most cited papers in the literature on agents, “Intelligent Agents: Theory and Practice” by Michael Wooldridge and Nicholas Jennings. In the end of this section we will look at some additional attributes.

2.5.1 Wooldridge’s and Jennings’ weak notion of agency

One of the most cited papers on software agents is Wooldridge and Jennings [1994], an adaptation of Michael Wooldridge’s Ph.D. thesis. The authors have a background in the AI tradition. It gives a good overview of theories, architectures and languages regarding agents. In the following we will examine their weak notion of agency, strip away its anthropomorphisms and present it in a more dry, software engineering sort of way. Their stronger notion of agency is treated in a later section. This is the weak notion of agency of Wooldridge and Jennings:

> **A weak notion of Agency**
>
> Perhaps the most general way in which the term agent is used is to denote a hardware or (more usually) software-based computer system that enjoys the following properties:
>
> **Autonomy**: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
>
> **Social ability:** agents interact with other agents (and possibly humans) via some kind of agent-communication language [Genesereth and Ketchpel 1994:a].
>
> **Reactivity**: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it.

Pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the Initiative.

[Wooldridge and Jennings 1994]

Let us examine these criteria, first one at a time and combined. The title of Wooldridge and Jennings [1994] is “Intelligent Agents: Theory and practice”, so for the sake of completeness, let us start with “Intelligence”. The stronger notion of agency is discussed in section 2.5.2.

2.5.1.1 Intelligence

There is no definition of “intelligence” in Wooldridge and Jennings [1994]. Lots of effort has been spent on arguing about definitions of the term, so maybe it is wise to leave it alone. Most authors seem to agree that, in the end, intelligence is in the eyes of the beholder. This is consistent with the notions in section 2.4. “Intelligence” is most useful in design and in describing user interfaces. A mainstream definition of “Intelligent Software” would be something like this:

Intelligent software is able to respond in a useful and consistent way to a variety of input. The larger the usefulness, consistency and variety, the more intelligent the software. For example, an intelligent personal assistant that can help me with booking travelling tickets *and* finding best prices for CDs would look quite intelligent. If it also updated my calendar in a way consistent to the travelling reservations it would look even more intelligent.

Another approach is from language: intelligent software communicates in a (reasonably complex) language and interacts in a way that is consistent with what it communicates.

For software, agent or not, as for humans, there is no sharp line between intelligence and non-intelligence. For most definitions of the term, intelligence is a continuum criterion for any system.

“Intelligence” defined in this way is not really useful to distinguish between different types of software. Software that adheres to its specification is intelligent in its own humble way.

2.5.1.2 Autonomy

Wooldridge and Jennings, [1994] had this definition of autonomy:

Autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state

All software, agent or not, is executed without direct human intervention by the underlying system as long as that system is up and running. “Operate”, on the other hand, can mean performing interaction with other software *or* with other humans than the original launcher of the program. If these “operations” are done without direct manipulation by the original launcher, it starts to look autonomous. The more autonomous in this sense, the more agent-like is the software. This criterion is sharp if it is fulfilled by one such interaction, and then a continuum from the simple to the complex with lots of autonomous interaction.

What about control over actions and internal state? Programs may act in the meaning “changing their environment”, and they may carry state but as far as control is concerned they shall adhere to their specification. For the most part, we do not want software to take any initiatives beyond the programmer’s intentions. “…control over their actions and internal state” does not make much sense when it comes to software, for the most part.

Another approach to autonomy is found in Artificial Life, cf. sections 2.5.3.7 and 8.2. Petrie, [1996] has a good discussion on “autonomy” of agents.

2.5.1.3 Social ability

The definition was:

> **Social ability:** agents interact with other agents (and possibly humans) via some kind of agent-communication language [Genesereth and Ketchpel 1994].

This quote is from a version of Genesereth and Ketchpel, [1994] we have not been able to obtain. In the one we did find, we find the following criterion:

> The criterion for agenthood is a behavioral one. An entity is a software agent if and only if it communicates correctly in an agent communication language like ACL. This means that the entity must be able to read and write ACL messages, and it means that the entity must abide by the behavioral constraints implicit in the meanings of those messages.

This is a sharp criterion. The ACL mentioned is developed within ARPA’s Knowledge Sharing Effort. ACL is a declarative inter-agent communication language. Note that a simple program that communicates with a small subset of ACL would also be an agent according this criterion alone. [Finin and Fritzson 1994],

2.5.1.4 Reactivity and Pro-activeness

> **Reactivity**: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;

This property is hard to distinguish from the autonomy of section 2.5.2

> **Pro-activeness:** agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the Initiative.

We can see a connection between the "…control over their actions and internal state" in the definition of autonomy and pro-activeness. If a computational entity changes its environment not only as a result of calls or messages from the outside, but as a result of its own evaluation of its inner state (in relation or not to the state of its environment), then it is proactive.

This is an extension of autonomy: The agent can act without first being stimulated. As Petrie [1998] points out, this is an important distinction as it states that an agent is something more than a server in the classical client-server model.

2.5.1.5 The Weak Notion's Criteria Combined

If we combine these four criteria, as Wooldridge and Jennings say we should, what do we have? With our own interpretations, this is what we get:

A software agent

- interacts without direct manipulation from the original launcher of the program with other agents or with humans
- communicates (with humans and/or other agents) in a (declarative) inter-agent communications language and functions in a way consistent with what it communicates.

This definition comes close to what Petrie [1998] calls typed-message agents and also to the definition of software agents of Genesereth and Ketchpel [1994:b]. We use the term speech-act agents as a common name for similar agents. More on speech-act agents follow in section 6.

2.5.2 Wooldridge’s and Jennings’ stronger notion of agency

After having presented their weak notion of agency Wooldridge and Jennings turn to what they call “a stronger notion of agency”. We have put the central part in **bold text.**

> For some researchers - particularly those working in AI - the term “agent” has a stronger and more specific meaning than that sketched out above. **These researchers generally mean an agent to be a computer system that, in addition to having the properties identified above, is either conceptualized or implemented using concepts that are more usually applied to humans.** For example, it is quite common in AI to characterize an agent using mentalistic notions, such as knowledge, belief, intention, and obligation [Shoham 1993]. Some AI researchers have gone further, and considered emotional agents [Bates 1994][Bates et al. 1992a]. (Lest the reader suppose that this is just pointless anthropomorphism, it should be noted that there are good arguments in favor of designing and building agents in terms of human-like mental states - cf. section 2.) Another way of giving agents human-like attributes is to represent them visually, perhaps by using a cartoon-like graphical icon or an animated face [Maes 1994] - for obvious reasons, such agents are of particular importance to those interested in human-computer interfaces.

[Wooldridge and Jennings 1994]

A software implementation does not consist of concepts, but of computer code. Normally we use a high level language like C, C++, Java, Simula or Smalltalk to create that code. The implementation is the code in the same way that a building is the physical thing you live in, not the drawings on which the construction once was based. When we have lunch we eat food, not recipes. Concepts that apply to humans and used to create computer code may be visible in the source code on high levels of abstraction, but not in the executables that result from compilation and optimization, or in the workings of interpreters (in the case of interpreted languages).

In the same way: the concepts used for conceptualizing (sic!) or designing the software may be visible on high levels of abstraction in the source, but not in the executable. If we reverse engineer the compiled code of an expert system we do not get high level models of “the world”, we get (the remains of) if-clauses.

Concepts usually applied to humans may be useful in modeling and design and they certainly show up in the experience of many people interacting with software. Can this stronger notion of agency be used to distinguish between different types of software? If we with software mean executables, we say no. If we mean software design or experiencing of user interface we may say yes. Using mentalistic notions or not in these phases seems to be a question of perspective, style and tradition.

2.5.3 Additional Attributes of Agency

2.5.3.1 Mobility

Mobile agents are pieces of executable code that as a part of their execution migrates from one address space to another. This is a working criterion for distinguishing between software. Mobility is however totally independent of agency as defined in section 2.5.1.5. A mobile agent can be, but does not have to be an agent according to 2.5.1.5, and an agent according to 2.5.1.5 can be, but does not have to be mobile. So when talking about mobile agents, we prefer not to call them “agents” only, but to keep the notion of mobility separate. More on mobile agents is found in sections 4 and 5.

2.5.3.2 Persistence

A program that is persistent over time looks more autonomous and thus more agent-like. If a program is up and running a week or a year after its launch, then it looks self-propelling and autonomous. This kind of autonomy is also a continuum. Many persistent programs like mailer-demons, mail-servers, WWW-servers do not qualify for agency. The notion of persistence alone does not capture the nature of “agency”.

Persistence may also refer to the ability to survive the environment. Most agent systems have some kind of persistent storage for its agents so that they can be restored after crashes or halted execution of the agent environment.

2.5.3.3 Learning, Adaptive

Learning software adapts to the environment and gets better at solving its assignment over time. A neural net is adaptive in this sense. The intuistic notion of “agent” does not seem to cover neural nets in themselves, but an agent that is also adaptive seems to reach a higher level of agency.

2.5.3.4 Acting on someone else’s behalf

An important property of many “agents” is “acting on someone else’s behalf”. In this category we find non-software like the secret agent, travel agent and the artist's booking agent. In our list of software agents from section 2.1.2, we have Assistant Agents and Programmed Participants as agents working on someone else's behalf. The User-interface Agents could possibly be placed in this category.

Some authors claim "acting on someone else's behalf" to *the* constituent factor of agency. In the 1998 summer edition of AI Magazine, a special issue on agents, Katia P. Sycara writes in her introduction:

> Agency was defined by Eisenhardt [1989] and mathematically modeled. *An agency relationship is present when one party* (the principal) *depends on another party* (the agent) *to undertake some task on the principal's behalf* (Our italics). The notion of agency covers cooperative coordination in MASs (multiagent systems) (agents depend on each other to perform their tasks); delegation in human interface design; object-orientated programming, where an object is using another; and self-interested coordination through contracting in MASs. [Sycara 1998], [Eisenhardt 1989].

This is close to being a means or instrument in the sense of MW3. Eisenhardt´s paper is often cited in literature on management theory. As "party", it seems, could refer to anything, this definition of "agency" is not very useful to discriminate between agent and non-agent software, or hardware. A word processor used to accomplish a task would turn into a word processing agent, a printer to a printer agent and so on.

2.5.3.5 Representing

In the category of "representing" in the sense of MW4 we have the federal or secret agent, business representative or the theatrical agent. They are all representatives of someone else. The Solicitor also belongs here. The difference between MW4 and MW3 is that in MW4 the agents interact with other agents whereas the MW3 agents just get deployed by their users. MW4 is thus a special case of MW3.

The agents in SICS MarketSpace discussed in section 7.3 would fulfill this property. These are also speech-act-based agents.

2.5.3.6 Impersonating agents –"Agency'" as Gestalt

By impersonating agents we understand software that project a believable persona on to the user/audience. The program appears to the user/audience as *someone*. Weizenbaum's Eliza is the classical example. Modern Elizas are even more believable.

Is “impersonating” a working criterion for software agency? Can we distinguish between impersonating software and non-impersonating software? By looking at the software itself, we can not. We can in many cases detect the intentions of the programmer and some software are better impersonators than other but personality is still experiential and belongs to Thoughts about Software.

In section 8 there are more on impersonating agents.

2.5.3.7 Artificial Life

In the discussion of autonomy of section 2.5 we stated that true autonomy does not make much sense when it comes to software, for the most part.

There is, however, an approach to software where real autonomy is desired and in some sense also achieved and that is Artificial Life. The idea is to construct, in a bottom-up fashion, the basic building blocks – the agents - and to give these the ability to breed, mutate and evolve. If proper feedback is built into the system the agents can adapt in an evolutionary way and solve a task with increasing efficiency. The Amalthaea information filtering system of the MIT Media Lab and the entertainment software Creatures of CyberLife Ltd. are two very different examples that are designed from this perspective. Cf. section 8.2 for more. [MIT 1999], [Creatures 1999],

2.6 Are Agents Going Out of Style?

Krogh [1996] has a list of agent research projects and commercial products and services that were attracting attention at the time. Some of them are still up and running, some are not. Some of them have stopped using the term “agent”, at least on their web-sites. From the examples on Krogh’s list these sites has dropped the use of “agents”: Firefly [1999], PointCast [1999] and My Yahoo [1999]. These sites still talk about “agents”: EUNett [1999], Verity [1999], Lotus [1999], Aglets Mobile Java agents, IBM [1998] and Banyan [1999].

Lotus uses the agent metaphor extensively. In Lotus Notes, a program for e-mail, calendar, group scheduling and so forth, the users can create “agents” that perform routine tasks like sending an automatic e-mail response when the user is on vacation. This is an agent at the user-interface level.

Scientific statements cannot be made from the short list of Krogh, but after having chased software agents all over the Internet for the last six month I have the impression that the usage of “agents” to sell programs like the Lotus Notes mail-answering facility is decreasing. This can be seen as a parallel to the decline of “intelligence” in software marketing since the peak of AI. Many companies nowadays keep their use of AI technology like neural nets, fuzzy logic or expert systems to themselves. [Waltzman 1998].

2.7 Summary

“Software agents” is a metaphor for autonomous software. Agents are software systems, or parts of software systems, which are in some sense autonomous. Many definitions of agency exist and many of them display a strong tendency toward anthropomorphizing the software. We have tried to provide criteria free from anthropomorphic metaphors to distinguish software agents from other software and to make distinct different types of software agents.

There is not *one* agent technique or type of software. The agent metaphor can be used on many stages of the software life cycle:

- In design to denote logical objects with some kind of autonomy. Example: Amalthaea info-mining system, section 7.4.
- In implementation to denote objects with agent characteristics. Example: mobile agents, section 5, speech-act agents, section 6
- In use of software to denote a user interface that is perceived as having a gestalt of its own. Examples: Eliza, section 8, Amalthaea, section 7.4, Creatures, section 8.2, Lotus Notes Agents, section 2.6.

Apart from having specific meanings, “agent” is a sales-word. It can be used to sell James Bond movies and detergents, as well as computer programs. Researchers also do have to sell their research projects to their superiors and to funding agencies.

Being a language police in the world of software is job for a linguistic Sisyphus; it is simply too much to do. However, because of the ambiguity of the terms “agent” and “agency” we recommend the uttermost care when using them as to avoid confusion. When other terms are as appropriate, use them instead, or use further specification. When you here about “agents” and “agent technology”, ask for clarification if the exact meaning is not provided by the context.

There are many valuable “agent techniques”. Some of them will be examined in following sections of this paper.

3 Distributed Computing

In this section we make an overview of distributed computing. Readers that are familiar with the field can browse through it and continue with the section on mobile agents. The objective of this section is not to give a full treatment of distributed systems but to put mobile agents into perspective by presenting how distributed computing differ from local computing, platforms for distributed computing and efforts made to standardize interoperability between distributed systems of different platforms.

3.1 Definition

We make a distinction between *local computing* and *distributed computing.* With local computing we mean programs that are confined to a single address space. Distributed computing refers to programs that make calls to other address spaces, possibly on other machines. [Waldo, Wyant, Wollrath, Kendall 1994]

Mobile agent technology is a special case of distributed computing.

3.2 Differences between distributed and local computing

A programming language or environment where function calls, objects and all other computational entities can be treated the same whether they are local - stored in the address space of the current process - or distributed – stored somewhere else – is said to have the property of *location transparency.* Location transparency has been said to be The Holy Grail of distributed computing. Location transparency makes writing distributed systems easy.

Whether a programming system is location transparent or not there are four areas where distributed computing is fundamentally different from local computing, namely latency, memory access, partial failure and concurrency.

3.2.1 Latency

Regardless of the increase of bandwidth of network connections a call on a remote object will for the most part be slower than on a local object, something which has to be considered when designing a distributed system.

3.2.2 Memory access

Pointer arithmetic cannot be permitted on a remote address space as the different address spaces are managed by two different operating systems instances. Allowing pointer arithmetics would be bad security and result in non-robust software. In C or C++ that could be a problem and C or C++ programmers might need to alter their style of programming when working on a distributed system. In Java, pointer arithmetic is not possible.

3.2.3 Partial failure

In distributed systems one has to cope with failure of parts of the system. Machines can crash and network connections can fail. Partial failure of the system must normally be treated at the application level. We need methods that throw exceptions and the programmer needs to keep track of calls to servers so that he can call others if responses are not received in due time. And he must also keep track of incoming answers from different servers. A server can answer too late, after a timeout.

3.2.4 Concurrency

In distributed systems there are concurrency problems as deadlocks, race conditions etc. [Waldo, Wyant, Wollrath, Kendall 1994]

3.3 RPC – Remote Procedure Call

With RPC the programmer can write code that issues an execution of a routine on another host. The idea is to make that call as *transparent* as possible to the programmer. This means hiding the communication between the calling client and the answering server from the programmer. In the classical client – server model the call to the distant procedure or function is actually a call to a local version of it, called the *client stub*. The client then blocks until the call returns. The client stub puts the parameters, if any, into a message and asks the kernel* to send it to the *server stub* bound to the actual server on the remote machine. The kernel passes the message to the server stub, which unpacks the message and calls the actual server with the delivered parameters. The server returns the result to its stub, the stub packs the result into a message and asks the kernel to deliver it to the client stub. The client stub unpacks the message and delivers the result to the original caller, which has been blocked since it performed its call. The packing and unpacking of parameters and results are sometimes called marshalling and unmarshalling.

The point with this long chain of calls and responses is that different aspects of the required computations are separated. The caller, or rather the linker, needs only keep track of where the client stub is. The client stub, on the other hand, can have a whole list of servers to chose from. If one server does not answer fast enough, it can ask another one. Or the system administrator can change the client stub so that a new and better server implementation is called. The server does not need to be bothered with who is calling it, or from where. It gets a call, looks at the parameters and returns a result, just as any (local) routine. [Tanenbaum 1992].

3.4 Object-Oriented Distributed Computing

Object-oriented distributed computing is about being able to call member functions on remote objects. This is sometimes called remote method invocation, even though this term nowadays mostly refers to Java RMI*, the Java Remote Method Invocation.

In a world of computer networks it is desired that objects written in different languages and in different styles can call methods on each other – provided of course that they have been given permission to do so. This can be achieved if calls are performed in a uniform way. Different efforts have been done to specify standards to achieve this aim, some of which are briefly presented in the following sections.

In later years distributed computing often include moving objects between address spaces. This is certainly the case when we get to mobile agent technology.

3.4.1 OMG – Corba

The main objective of the Object Management Groups Corba effort is to establish standards of interoperability between distributed systems of different platforms.

3.4.1.1 The Object Management Group – OMG

The Object Management Group is a non-profit consortium created in 1989 with the purpose of promoting theory and practice of object technology in distributed computing systems. In particular, it aims to reduce the complexity, lower the costs, and hasten the introduction of new software applications. Originally formed by 13 companies, OMG membership grew to over 500 software vendors, developers and users.

OMG realizes its goals through creating standards which allow interoperability and portability of distributed object oriented applications. They do not produce software or Implementation guidelines; only specifications which are put together using ideas of OMG members who respond to Requests For Information (RFI) and Requests For Proposals (RFP). The strength of this approach comes from the fact that most of the major software companies interested in distributed object oriented development are among OMG members. [Keahey 1998]

3.4.1.2 The Object Management Architecture – OMA

OMA is OMG's high level vision of a complete distributed environment. It consists of four components that can be roughly divided into two parts:

- System-oriented components
  - Object Request Brokers
  - Object Services
- Application-oriented components
  - Application Objects
  - Common Facilities

The Object Request Broker is the foundation of OMA and manages all communication between its other components. It allows objects to interact in a heterogeneous, distributed environment, independent of the platforms on which these objects reside and techniques used to implement them. In performing its task it relies on Object Services which are responsible for general object management such as creating objects, access control, keeping track of relocated objects, etc. Common Facilities and Application Objects are the components closest to the end-user, and in their functions they invoke services of the system components.

3.4.1.3 The Common Object Request Broker – CORBA

CORBA is a specification of a system that provides interoperability between objects in a heterogeneous, distributed environment in a way transparent to the programmer. Its design is based on OMG Object Model. In this model clients request services from objects – also called servers – through a well defined interface, specified in OMG IDL – Interface Definition Language. (Figure 1).

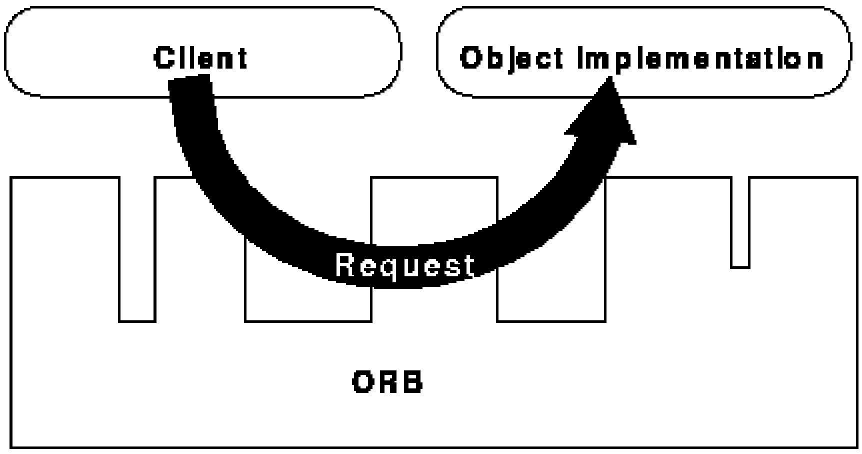


Figure 1.
The calling client reaches the object implementation via the Object Request Broker. [OMG 1998]

The request is an event that carries information including the object reference, the name of the operation and arguments, if any. The object reference is a unique identifier. [OMG 1998], [OMG 1999]

3.4.2 Microsoft and DCOM

Microsoft is a member of OMG, but they still have their own model of distributed computing – DCOM:

> The Distributed Component Object Model (DCOM) is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. Previously called "Network OLE," DCOM

> is designed for use across multiple network transports, including Internet protocols such as HTTP. DCOM is based on the Open Software Foundation's DCE-RPC spec and will work with both Java applets and ActiveX® components through its use of the Component Object Model (COM). For example, a developer could use Java to build a Web browser applet that calculates the value of a portfolio of securities, using DCOM to communicate stock values to the applet in real time over the Internet. [Microsoft 1999]

3.4.3 Java RMI and Serialization

Remote method invocation means calling a method in a remote object. Sun Microsystems, the creator of the Java programming language, calls their Java-to-Java distributed computing technology RMI - Remote Method Invocation. We should thus be aware of some potential confusion about this term. When we hear it today it mostly refers to Java technology, not remote method invocation in general.

> Remote Method Invocation (RMI) enables the programmer to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. A Java program can make a call on a remote object once it obtains a reference to the remote object, either by looking up the remote object in the bootstrap naming service provided by RMI or by receiving the reference as an argument or a return value. A client can call a remote object in a server, and that server can also be a client of other remote objects. RMI uses Object Serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism.
>
> ---
>
> Object Serialization extends the core Java Input/Output classes with support for objects. Object Serialization supports the encoding of objects and the objects reachable from them into a stream of bytes and it supports the complementary reconstruction of the object graph from the stream. Serialization is used for lightweight persistence and for communication via sockets or Remote Method Invocation (RMI). The default encoding of objects protects private and transient data, and supports the evolution of the classes. A class may implement its own external en-

> coding and is then solely responsible for the external format. [Sun 1999]

Sun has declared their intention to make Java RMI CORBA-compliant, including support for OMG's new IIOP – Internet Inter Orb Protocol.

3.4.4 Reaching the Holy Grail: Oz – Mozart

SICS – the Swedish Institute of Computer Science – in Stockholm, Sweden, has developed a programming language called OZ and a programming system called Mozart that implements Oz. The Mozart implementation was done in co-operation with researchers from Programming Systems Lab, DFKI, Universität des Saarlandes, and Collaborative Research Center in Germany and Departement d´Ingénierie Informatique at Université catholique de Louvain, Belgium.

> The Mozart system provides state-of-the-art support in two areas: open distributed computing and constraint-based inference. Mozart implements Oz, a concurrent object-oriented language with dataflow synchronization. Oz combines concurrent and distributed programming with logical constraint-based inference, making it a unique choice for developing multi-agent systems. Mozart is an ideal platform for both general-purpose distributed applications as well as for hard problems requiring sophisticated optimization and inferencing abilities. We have developed applications in scheduling and timetabling, in placement and configuration, in natural language and knowledge representation, multi-agent systems and sophisticated collaborative tools. [Mozart 1999]

The Oz/Mozart system provides total location transparency for all computational objects. Oz is an interpreted language and all management of locality, sharing of variables, objects etc. is handled by the interpreters*, the Mozart VMs. Thus Oz/Mozart acts like a kind of middleware that presents an API with a convenient level of abstraction, relieved from OS-specific details. From the Introduction to the on-line tutorial:

> The Mozart system supports network-transparent distribution of Oz computations. Multiple Oz sites can connect together and automatically behave like a single Oz computation, sharing variables, objects, classes, and procedures. Sites disconnect automatically when references between entities on different sites cease to exist.

> In a distributed environment Oz provides language security. That is, all language entities are created and passed explicitly. An application cannot forge references nor access references that have not been explicitly given to it. The underlying representation of the language entities is inaccessible to the programmer. This is a consequence of having an abstract store and lexical scoping. Along with first-class procedures, these concepts are essential to implement a capability-based security policy, which is important in open distributed computing.

As the system is completely location transparent, mobility is trivial. From what we have learned from people that have used Oz, it is a fantastic platform for distributed computing, but its learning curve is somewhat steep. The system is available for downloading in UNIX, Windows95/NT and Linux versions at Mozart, [1999:b].

4 Mobile Agent Systems

4.1 General Aspects

4.1.1 Definition

The term “mobile agent” as used by the providers of systems like ObjectSpace Voyager and IBM Aglets refers to a computer program or a piece of computer code that can

- start executing on one host and during that execution and as a part of that execution,
- suspend its execution,
- conserve its state, possibly including program counter and CPU registers,
- move itself (with the help of the underlying infrastructure) to another host,
- unpack itself,
- restore its state and
- resume execution on the new host.

We must be aware that this definition refers only to the mobility of the code, not to the “agency” of the software in the sense of our definition of section 2.5.1.5. Platforms like ObjectSpace Voyager and IBM Aglets make it easy to build distributed applications in the “Mobile Agent Paradigm”, but the intelligence and thus most of the “autonomy” must be provided by the developers.

4.1.2 Basic Model

The basic concepts concerning mobility in the model of General Magic's Telescript Language are *place* and *agent.* Both concepts represent individuals or organizations in the physical world. The places are stationary and agents are mobile or stationary. Mobile agents can move from one place to another. Agents can communicate with other agents with messages regardless of location, provided the place supports the communication. [Mchacko 1999].

Most mobile agent systems have abided close to this model.

4.1.3 Security

Mobile agent security has two aspects, protection of the hosts with the agent execution environment from attacks from malignant agents and the protection of the mobile agents themselves from attacks from other agents and from the hosts on which they execute.

4.1.3.1 Protecting Hosts

There are several models when it comes to protecting hosts from malignant mobile code.

The Sandbox model, where the idea is to contain the mobile code in such a way that it cannot cause any damage to the executing environment. This usually involves restricting access to file system and limiting the ability to open network connections. The most widespread implementation of a sandbox is the Java interpreter inside Internet browsers.

Code Signing, where the clients manage a list of entities they trust. When a mobile executable is received, the client verifies that it was signed by an entity on the list. If so, then it is run, most often with all of the user's privileges.

Hybrids: Sandboxes and Signatures, where a combination of Sandbox and Code Signing is used. Cf. the following section on Java.

Firewalling, where you decide whether or not to let the code run on your hosts when it enters your domain. Commercial solutions exist, for example Finjan Software and Security 7. This approach is seriously limited. The halting problem states that there a general algorithm does not exist that can predict the behavior of an arbitrary program. [Rubin and Geer 1998]

4.1.3.2 Protecting Agents

Protecting agents from hosts is in principle very difficult. The host has full access to the code and can reverse-engineer it if desired. Using encryption can solve some security problems, but when a mobile agent for example has to compare prices on an untrusted host the host can see which price is lowest even if it cannot see who provides it. One way of achieving some privacy is to keep a resident agent on a *smart card,* installed on the foreign host. Data can be carried in encrypted form and all computation requiring privacy can be done onboard the smart card. That is almost like having your own host in a remote location. In a business-to-consumer situation for instance, this is normally not possible.

4.1.4 Persistence

Most agent platforms provide persistent storage of their agents. Otherwise server crashes would end the autonomy of the individual agents. The underlying system should be able to store the agent with its state when the agent is not interacting with any other agent or with the execution environment and bring the agent “back to life” when necessary.

4.1.5 Advantages

Many applications where mobile agents are a good, convenient solution can readily be implemented using well-known techniques as RPC or remote method invocation (basically synchronous communication) or message passing (basically asynchronous communication). Several advantages with mobile agents however, make them a preferable foundation for distributed systems.

- **They reduce network load:** Mobile agents consume less network resources by moving the computation to the data rather than the data to the computation. In traditional distributed systems many interactions between distant machines are often needed to accomplish a certain task, especially when security is an issue. With mobile agents it is possible to package a conversation and dispatch it to a distant host where communication can take place locally. While it is true that lack of bandwidth is technically no problem, there will always be situations where minimization of use of bandwidth is desired.
- **They overcome network latency:** In a real-time system like an industrial robot you can use mobile agents to push instructions on to the robot. To use communication on a local network could well be too slow in a time-critical application.

- **They make disconnected operation possible:** Mobile agents do not require a continuous connection between machines. This gives higher fault tolerance and it gives the end-user access to the full computational resources of his/hers machine after launching the mobile agent.
- **They can give support for weak clients:** A weak host (slow CPU, low bandwidth connection) gets good help from being able to package a computation or network interaction in a mobile agent.
- **They can encapsulate protocols:** Mobile agents make it easy to maintain a distributed system. If you need to alter an implementation you can simply call back the old agents and launch new ones.
- **They facilitate hardware maintenance:** When a server needs to be taken down all agents can be warned in a similar way and then decide on their own if they want to go somewhere else, get stored locally or die.
- **They offer possibilities for customization:** Mobile agents allow clients and servers to extend each other's functionality by programming each other.
- **They offer a convenient programming paradigm:** Mobile agents hide the communication channels but not the location of the computation. Once the mobile agent servers are in place across the network, it is easy to build sophisticated distributed systems.

[Chess, Harrison and Kershenbaum 1995], [Dartmouth 1997], [Lange and Oshima 1998], [Lange and Oshima 1999].

4.1.6 Critical Aspects

Regardless of how a mobile agent system is implemented there are some issues that have to be dealt with:

- **The execution environment has to be standardized.** This is no more remarkable than that you have to agree on a protocol in a distributed system, but it has to be remembered when designing an "open system" to which you want to attract many players. In some cases it may be easier to use stationary agents and message passing only.
- **Security.** As already pointed out, you have to protect both your host and your agents. Protecting hosts is most immediately apparent, but protection of agents seems to the most difficult in

the long run – it is in principle impossible for an agent to keep its computations concealed from the executing host.

For more on mobile agent security, cf. for example Cheng, [1997], for more on security in Java-based systems, cf. section 4.2.2.

4.2 Existing Mobile Agent Systems

In this section we have compiled some interesting systems for mobile agents. For additional information, cf. for example Banks, [1997] and W3C [1996].

4.2.1 Non Java Systems

4.2.1.1 General Magic - Telescript

The company General Magic claims they invented mobile agent technology. They developed an interpreted object-oriented language called Telescript to implement mobile agents. Wooldridge and Jennings acknowledge Telescript to be "…perhaps the first commercial agent language." [Wooldridge and Jennings 1994].

The model of Telescript builds on the concepts *agent* and *place*. Places are virtual locations where agents can meet and interact. The focus when Telescript was developed obviously was Electronic Commerce. The idea was that *agents* representing vendors and consumers could meet in an electronic market-*place* and perform spontaneous electronic commerce. The Telescript agents could carry a virtual currency called Teleclicks.

> A network equipped for mobile agents makes possible an electronic marketplace in which the agents of providers and consumers of goods and services can find and interact with each other. New forms of electronic commerce and community will result. [Mchacko 1999].

Around 1996 General Magic started to reimplement their mobile agent technology in Java under the name Odyssey. General Magic does not provide information on Telescript. The Telescript Language Reference v.1.0 is available at Mchacko [1999]. [Gardner 1996], [Tham and Friedman 1996], [White 1996], [Moore 1998], [Jackson], [Agent Society 1997:a], [Agent Society 1997:b], [Agent Society 1998].

4.2.1.2 Agent Tcl

Dartmouth Collage has (or possibly had) a project on mobile agents called Agent Tcl with their own programming language, Tcl*. It has extensive navigation and communication services, security mechanisms, and debugging and tracking tools. The agents have the ability to migrate at any point in their execution taking their entire state with them to their new location. [Dartmouth 1997], [Dartmouth 1998]

4.2.1.3 Ara

University of Kaiserslautern, Germany, has also used Tcl for their mobile-agents project. [Kaiserslautern 1997]

4.2.1.4 Tacoma

The mobile agent project of University of Tromsø, Tacoma, (Tromsø And Cornell Moving Agents) focuses on operating system support for agents and how agents can be used to solve problems traditionally addressed by other distributed computing paradigms, e.g. the client/server model. The latest version of Tacoma, TACOMA Version 1.2 is based on UNIX and TCP. The system supports agents written in a variety of programming languages (C, Tcl/Tk, Perl, Python, and Scheme). TACOMA 1.2 is implemented in C. [Tacoma 1999], [Lange and Oshima 1998]

4.2.2 Java Systems

The advent of Java – The Language of the Internet – has brought about an even bigger interest in mobile agents. There are a number of Java platforms for mobile agents available. All of them provide a basic infrastructure for mobility and messaging between agents, some of them also include more advanced functionality. The "higher" functionality of autonomous or intelligent agents is up to the user of the platform to provide.

We have taken a closer look at two of the Java platforms, ObjectSpace Voyager and IBM Aglets. A list of other systems is presented below with less or nothing more than the information provided by the manufacturers.

4.2.2.1 Java Advantages

Java has some features that make it a convenient platform for secure and robust mobile agent systems.

- **Platform independence:** The Java Virtual Machine, JVM, provides everything needed to build an engine for places as well as mobile agents.

- **Security:** Java is designed for the Internet and many security issues were taken in consideration when designing the language. For a discussion on Java and security, cf. section 4.4.2.2.
- **Sockets:** It is easy to set up a TCP/IP connection in Java. Two classes are involved, Socket and ServerSocket. Communication over such a connection is as simple as file I/O.
- **Dynamic class loading:** The JVM can define and load classes at runtime. A protective namespace can be provided for each agent, which therefore can execute safely and independently of each other. Classes can be loaded via the network.
- **Reflection:** With reflection, Java programs can obtain information about the fields, methods and constructors of loaded classes in order to be able to operate on the instance objects of these classes, all within security restrictions. Thus agents can be smart about themselves and other agents.
- **Serialization:** Serialization in Java is a convenient way to send objects from host to host. When serializing an object, all other objects reachable from it are serialized as well. Thus it is possible to restore the entire graph at a later time. This feature is used for migrating as well as for persistent storage.
- **Threads:** Threads are "lightweight" processes that can coexist within the same JVM instance. Threads make possible the independent execution of several agents within the same place. Java also has a number of synchronization primitives that makes agent interaction easier.
- **Low learning curve:** Most young programmers of today know Java. Not having to learn a new language gives shorter time-to-market.

[Lange and Oshima 1998], [Ethington 1998]

4.2.2.2 Java Security

Java was designed with security in mind.

- It is impossible to do pointer arithmetic and thus impossible to overwrite data in the way you can in C or C++.
- Typecasting is restricted.
- Basic semantics of the language cannot be violated even by tampered code.

- JDK 1.2 uses a sandbox/code signing hybrid security approach where all classes – local, remote, signed or unsigned – are subject to access control decisions. A security policy defines the access each piece of code has to resources on the client. Signed code can run on different privileges based on the key used.
- The class loader converts remote bytecode into data structures representing Java classes. Any class loaded from the network requires an associated class loader that is a subtype of the Classloader class. This means that the only way to add remote classes to a machine's local class hierarchy is via the class loader.
- The Verifier performs static checking on the remote code before it is loaded. It checks that the remote code
  - Is valid virtual machine code.
  - Does not overflow or underflow the operand stack.
  - Does not use registers improperly.
  - Does not convert data types illegally.

  These checks attempt to verify that remote code cannot forge pointers or access arbitrary memory locations. This is important because if an agent or applet could access memory in an unrestricted fashion, it could run native machine code on the client – an ultimate hacker goal and the definition of disaster.
- The Classloader classifies operations as safe or potentially harmful. Safe operations are always allowed, but potentially harmful ones cause an exception and defer a decision to the security manager. In effect, the Security Manager classes represent a security policy.

[Lange and Oshima 1998], [Rubin and Geer 1998].

4.2.2.3 IBM Aglets

In the taxonomic tradition of applets and servlets, IBM has created the Aglets platform. It provides agent administration and transportation infrastructure and all the baseclasses needed to build mobile agent applications. [IBM 1998].

4.2.2.4 General Magic Odyssey

General Magic claims they invented mobile agent technology (cf. section 4.2.1 and 4.4.1.1). First implemented in their own language Telescript, General magic has now reimplemented their platform in Java under the name of Odyssey. The Odyssey class library provides everything needed to build a mobile agent system. [General Magic 1998:a]

4.2.2.5 Mitsubishi Concordia

American Mitsubishi has built a mobile agent system with sophisticated security and messaging capabilities. [Mitsubishi 1999]

4.2.2.6 ObjectSpace Voyager

The Voyager system of Dallas-based ObjectSpace is a strong candidate for a general platform for distributed computing in Java. A closer look at Voyager is presented in section 5.5. [ObjectSpace 1998]

5 Mobile Agents Test Case: DiSolver

Some researchers take the position that mobile agents is a solution still looking for its problem. An application where mobile agents *is* an adequate technique would be one where moving a computation from one machine to another is the essential part of the application. We designed the DiSolver system for distributed problem solving for two reasons

- To give an example where mobile agents solve a problem,
- To have a test case for comparing the two mobile agents platforms ObjectSpace Voyager and IBM Aglets.

The source code of the DiSolver implementations on ObjectSpace Voyager and IBM Aglets can be found at http://www.sisu.se/~jonas/ThesisSource.doc

5.1 DiSolver – Grassroots Parallel Computing

The basic idea with DiSolver is to use mobile agents to be able to move a computation from one machine to another *or* automatically distribute a CPU-intensive computation onto several machines. The immediate advantage is twofold:

- The original machine is relieved of some load and can do other work.
- The computation may be completed faster.

There are several different scenarios in which a DiSolver-like system could be used. Most personal computer- and workstation CPUs are idle most of the time. With a DiSolver-like infrastructure for distributed computing you can let your spare CPU time out on hire on an open market. If you have proper access to the priority-queue, you can let programs from outside come in and execute with a low priority, and you would hardly notice they were there. Payments could be handled with a micropayment system.

The same infrastructure can be used to turn the whole of Internet into a super parallel computer. Suppose you have a big computation that can be split up into two smaller ones, which in their turn can be solved separately and whose solutions can be assembled to the solution of the original problem. Put the problem splitting- and assembling definitions into a general distributed problem-solver – a DiSolver - give it a list of free CPUs and launch it on the net. In due time, it comes back to you and delivers the result.

A corporation can use the same idea. Instead of buying expensive supercomputers, big computations can be run on the in house desktops, either at night when the employees are at home, or in the background with low priority, all safely behind the corporate firewalls.

5.1.1 Splitting Problems and Assembling Solutions

For the DiSolver to work in the simplest case we need a problem that can be easily split up into two sub-problems which in their turn can be solved separately. Furthermore we have to be able to combine these two solutions to form the solution of the original problem. We call an easily split problem. Far from all problems are easily split. In some cases where an easy split is impossible it might still be possible to construct a DiSolver-like system if problem instances can communicate. This is far more complicated to generalize so we leave it out in this discussion. The problem used in our test case is the simplest imaginable: addition of lists of integers.

5.2 DiSolver: Corporate Version – Outline

In this section we describe the DiSolver system of distributed computing set in the context of a company running easily split computations on the in house desktops and workstations.

The system consists of the following parts:

- **The mobile agent execution environment.** In our test case this would be the IBM Aglet Server or the Voyager server. The mobile agent execution environment is installed on the machines comprised by the DiSolver system.

- **CPU Brokering Managers.** Stationary agents residing at the agent servers of each host in the system. Keep lists of hosts available to the system on which to distribute.

If someone wishes to perform a big computation and would like other CPUs to help him out, he defines how one instance of a problem shall be split up into two smaller ones, how these shall be computed and finally how the sub-solutions shall be put together to the actual solution of the original problem. The splitting factor does not have to be two, even if that is the easiest to implement.

The definitions are put into a DiSolver agent together with the problem instance to be solved. The DiSolver requests a list of presumably available CPUs from the local CPU Brokering Manager, splits the problem-instance up into smaller ones, puts them into new instances of DiSolvers and sends them off to the new hosts.

Once there a child DiSolver looks at her problem instance and decides whether it is small enough to be computed locally. If it is, she does so and sends the result back to her parent to be assembled; if not she repeats what her parent just did before her.

5.2.1 Central Algorithm

```
If smallEnough (problem)
        solution = computeLocally (problem)
else
{
        problemList = partition (problem)
        solutionList = computeDistributed (problemList)
        solution = assemble (solutionList)
};
return solution;
```

5.3 Design: System

In this section we present the design of the system

5.3.1 Environment

The Environment is the execution environment of the mobile agents. It also comprises facilities for messaging between agents.

5.3.2 Abstract class Problem

The class Problem contains the problem instance at hand. The essentials are the abstract classes that have to be implemented by the user of the system. In our implementations addition of integers the actual problem is put in a subclass to Problem, in our case in Addition.

5.3.2.1 Attributes

No Attributes.

5.3.2.2 Operations

Problem ()

Constructor of class Problem.

Solution: computeLocally () (abstract)

computeLocally computes the solution of the problem instance of this Problem.

Problem: partition () (abstract)

partition splits up the problem instance of this problem into two by returning a Problem-object containing “half the problem instance” of this problem and leaving the other half in this problem.

Integer: size () (abstract)

size returns the size of the problem instance as an integer.

Boolean: tooBig () (abstract)

tooBig returns true if the problem instance of this Problem is too big to be computed locally.

5.3.3 Abstract class Solution

The class Solution carries solutions to the problem at hand.

5.3.3.1 Attributes

No attributes.

5.3.3.2 Operations

assemble (Solution s) (abstract)

assemble assembles the solution in this Solution with the solution it takes as an argument.

display () (abstract)

Display writes the solution to std out.

5.3.4 DiSolver

The class DiSolver is the mobile agent of the system. The two implementations do not follow this design exactly, due to specifics of the platforms.

5.3.4.1 Attributes

Vector: problemList

Storage for Problems during partition.

Vector: solutionList

Storage for Solutions that has arrived from children DiSolvers.

Vector: nodeList

List of available nodes.

Boolean: isTop

isTop is true if this is the root DiSolver.

DiSolver: parent

Parent is the parent of this DiSolver.

5.3.4.2 Operations

Public:

run (Problem p)

The main method that performs all the work of this DiSolver instance: solves the problem and returns the solution to the caller.

receiveSolution (Solution s, Integer index)

receiveSolution stores the solutions delivered by children to this DiSolver and assembles them when all have arrived. The index refers to where in the solutionList s is to be placed.

Private:

computeDistributed (problemList, nodeList, solutionList)

computeDistributed distributes the list of partitioned problems onto available CPUs.

partition (problem p)

Splits a problem instance up in two (or more) smaller problem instances by calling the partition operation of the class Problem.

Solution: assemble (solution s1, solution s2)

Assembles two (or more) solutions into one big one by calling the assemble operation of class Solution.

handleSolution (Solution s)
Delivers the computed solution to the parent.

5.4 Design: Addition of Integers

The application we chose for testing the DiSolver system is the simplest imaginable: Addition of integers. It involves two classes, Addition and Sum, that inherits from Problem and Solution respectively.

5.4.1 Addition (extension of Problem)

Addition holds a problem instance of addition of integers as a list of integers.

5.4.1.1 Attributes

Vector v
Holds a list of Integers.

5.4.1.2 Operations

All the abstract operations in Problem are implemented in Addition.

Solution: computeLocally ()
computeLocally adds the integers in v and returns the result as on object of class Sum.

Problem: partition ()
partition splits up v in half, leaves one half in this Addition and puts the other half in a new instance of Addition and returns it as a Problem.

Integer: size ()
size returns the size of v.

Boolean: tooBig ()
tooBig returns true if the size of v is bigger than four.

5.4.2 Sum (extension of Solution)

The class Solution carries solutions to the problem at hand.

5.4.2.1 Attributes

Integer val
Holds the value of this Sum.

5.4.2.2 Operations

assemble (Solution s)

assemble adds the val of s with the val of this Sum and assign it to this val.

display ()

display writes val to std out.

5.5 Evaluation of two Java Mobile Agent Platforms

We have taken a closer look at two Java-based platforms for mobile agents, the IBM Aglets system, v.1.1b and ObjectSpace Voyager, v.2.0, by implementing the DiSolver system for "grassroots parallel computing", a test-case application for mobile agents.

5.5.1 Similarities

Both platforms are written in 100% pure Java and consist of class libraries plus documentation. The architectural solutions are similar. An agent server is the central part in both systems. The overall model of both is similar to the one originally presented by Telescript. *Agents* inhabit *places.* Agents can migrate between places and send messages to each other. Messaging is primarily asynchronous. Agents are not accessed directly but through "virtual references" (Voyager) or "proxies" (Aglets). This supports location transparency and security and makes it possible to forward messages automatically when an agent moves from one place to another.

5.5.2 IBM Aglets v.1.1 Beta

Agents in IBM Aglets are subclassed from the Aglet class. The aglet class has a number of abstract functions that the agent environment calls to control what code is executed at major events of the agents' life cycle. These major events are *Creation* (creation or cloning), *Disposal, Mobility* (dispatch and retract) and *Persistence* (deactivate and activate). The functions connected to these events are onCreation (), onCloning (), onCloned (), onDisposal (), onDispatching (), onArrival (). These functions are called by the environment if the agent calls its constructor, Clone () (onCloned () is called when the new identical agent is brought to life), dispose () and dispatch () respectively. onArrival () is called by the environment when an agent has arrived on a new location. Event listeners have to be added to the agent by the programmer in order to register the right function with the environment.

This plethora of functions are easier to manage than it may seem in this presentation, but never the less, it makes programming with Aglets quite different from local Java programming.

Incoming messages are all handled by the handleMessage (Message m) method that has to unpack the message to decide what local function to call with the possibly attached arguments, cp. the server stub in section 3.3 on remote procedure calls.

In the Aglet 1.1 beta implementation that we tested the agents are created and managed through an application program called Tahiti. It works as an agent server and provides a graphical user interface for monitoring agents migration and so forth.

Good documentation on IBM Aglets is provided in Lange and Oshima [1998]. This source also provides material on mobile agents in general and particularly on mobile agents in Java. [IBM 1998], [Lange and Oshima 1998].

5.5.2.1 DiSolver Implementation

We could not access the in other ways than through the Tahiti server. To define the problem instance and launch the root DiSolver we could have used a special stationary agent. That would have given a cleaner code in class DiSolver. Instead we put the setProblem inside DiSolver.

The implementation always delivered the correct result, but did not work cleanly. Exceptions were thrown that we were unable to track down and different runs of the program did not result in exactly the same behavior in agent migration. The code is included in appendix.

5.5.3 ObjectSpace Voyager v.2.0

Voyager is a general platform for distributed applications in Java.

Complete bi-directional Corba integration. ObjectSpace provides an extra Java package that turns Voyager into a Corba 2 client or server. Thus a Voyager object can communicate with any Voyager or Corba server and vice versa.

Any class can be remote enabled without modifying its code. The user need not bother about location of the .class-file as long as it is in the classpath of the Voyager server.

Messages can be sent to remote objects using regular Java syntax. Unlike in Aglets, the agent environment does not call any mandatory functions at certain stages of the agent lifecycle. This also makes programming in Voyager more like regular Java programming.

Scalable group communication. Voyager has a group communication facility based on *spaces* and *subspaces*. A space can consist of one or more subspaces. When a message is sent to one of the subspaces in a given space it is first replicated and sent to all other subspaces in that space before being delivered to the members of the local subspace. This results in rapid parallel distribution. This group communication functionality scales easily.

Support for key Java RMI APIs

Voyager v.2.0 was given away without restrictions. Current versions available for download online are allowed for internal use only.

The current version (April 1999) is v. 2.0.2. ObjectSpace have declared support for DCOM in second quarter of 1999.

ObjectSpace provide excellent documentation online.

[ObjectSpace 1999]

5.5.3.1 DiSolver implementation

We implemented DiSolver as a Voyager agent and as an ordinary remote enabled class. Voyager is easy to work with and executes cleanly with no strange exceptions appearing. It requires a driver to instantiate the root DiSolver. The driver also contains the definition of the problem to be solved. The agent version of the code is included in appendix.

5.5.4 Comparison

When comparing the two platforms Voyager comes out as the superior. It is cleaner in design, more location transparent, executes cleaner, has better documentation and better interoperability with other systems for distributed computing. The ObjectSpace web site also proudly presents a line of awards and good reviews for the Voyager platform. We have not tested security issues.

6 Speech-Act Agents

6.1 Introduction

The intelligent agents that Wooldridge and Jennings describe in their soft notion of agency bear a strong heritage from the AI tradition in general and from the ARPA Knowledge Sharing Effort (KSE) in particular. Genesereth and Ketchpel [1994] give this definition of this type of software agent:

> The criterion for agenthood is a behavioral one. An entity is a software agent if and only if it communicates correctly in an agent communication language like ACL. This means that the entity must be able to read and write ACL messages, and it means that the entity must abide by the behavioral constraints implicit in the meanings of those messages.

[Genesereth and Ketchpel 1994]

The ACL (Agent Communication Language) mentioned in this quote refers to the one developed in the ARPA Knowledge Sharing Effort. This ACL implements a communication model based on speech-act theory. Speech-act theory was first described by Austin, see e.g. Austin [1962]. A speech-act-based language in this context consists of on outer language describing *performatives* and an inner language describing *content*. Performatives are describing the intention of communicating the content. In an example from natural language a sentence like “I promise you a glass of beer” the performative is “promise” and the content is “a glass of beer”, while “I” is the sender and “you” the receiver. In order to make this model of communication complete we also need *ontologies* defining the semantics of the inner and outer languages, in this example as to define what is meant by “promise” and “a glass of beer” respectively. Gruber gives this short definition of ontology:

> An ontology is a specification of a conceptualization.
> ----
> In the context of knowledge sharing, I use the term ontology to mean a specification of a conceptualization. That is, an ontology is a description (like a formal specification

> of a program) of the concepts and relationships that can exist for an agent or a community of agents.

[Gruber 1999]

Note that the inner and outer languages of a speech-act based agent communication language are not designed as programming languages. Speech-act based agents can be implemented in any high level language. The Speech-act based ACL defines how the agents communicate. We use the term speech-act agents (SA-agents) as a common label for software agents in this tradition. Charles Petrie of the Stanford University Center for Design Research (CDR), comes close with his definition of his typed-message agents:

> We follow Genesereth's approach, but differ somewhat from the definition of this paper in light of our experience with Next-Link agents and comparison with other KQML-like agents, our Typed-Message Agents are defined in terms of communities of agents. (We may also call these "ACL Agents" after Genesereth.) The community must exchange messages in order to accomplish a task. They must use a shared message protocol, such as KQML, in which the some of the message semantics are typed and independent of the application. And semantics of the message protocol necessitate that the transport protocol not be only client/server but rather a peer-to-peer protocol. An individual software module is not an agent at all if it can communicate with the other candidate agents with only a client/server protocol without degradation of the collective task performance.

Petrie [1998]

The notion of autonomy has an additional meaning when applied to speech-act agents:

> …an agent may not constrain another agent to perform a service unless the other agent has advertised its willingness to accept such a request

[Genesereth and Ketchpel 1994]

This is different from the client/server model where the server delivers on every call.

SA-agents do not necessarily have to be very “intelligent”. They merit for agency by abiding to the semantics of their communication and that language can be very small and limited. But what if alleged SA-agents cheat? In a closed system where the programmer has full control over his/hers agents that of course is not a problem. Lack of veracity in such a system is a bug (Cp. the theodicy* problem in monotheism!). But in an open system as the trading system described in section 7.3 there are always devils and veracity is an issue. Trading agents committing to contracts and later not fulfilling their obligations will cause problems. There has been research in this area using results from game theory. As such an open system looks pretty much as real life it seems likely that some kind repression mechanisms towards cheating agents will evolve. Cf. the discussion on “Prisoners Dilemma” in Dawkins [1976].

6.2 Agent Communication Languages

6.2.1 ARPA’s Knowledge Sharing Effort

The Knowledge Sharing Effort (KSE) was launched by ARPA* to create infrastructure for distributed knowledge-based systems. The idea was that a new knowledge-based system should be able to use knowledge and reasoning capabilities of existing systems, thus sharing knowledge, problem solving techniques and reasoning services. This work resulted in the specifications for the Knowledge, Query and Manipulation Language (KQML) and Knowledge Interchange Format (KIF). The current status of KSE as a research project is unclear. The referred web-site has not been updated since 1994. [Knowledge Sharing Effort 1994].

6.2.2 KQML and KIF

KQML, Knowledge Query and Manipulation Language, is an Agent-Communication Language designed to support run-time knowledge sharing among intelligent systems, or agents. It is both a message format and a message-handling protocol. KQML comprises an extensible set of performatives that express the permissible operations that agents can perform on each other’s knowledge. On this substrate higher-level agent interaction such as negotiating, contracting, cooperative problem solving etc. can be built.

KQML has been used inside the scope of KSE and elsewhere to prototypes and testbeds in concurrent engineering, intelligent design and intelligent planning and scheduling.

A KQML performative takes as one of its parameters an expression written in a content language, e.g. KIF.

6.2.2.1 Examples of performatives

- **Tell**
  Indicates that the content statement of the **tell** performative is in the knowledgebase of agent performing it.

- **Deny**
  Indicates that the content of the **deny** performative is not true to the sender

- **Error (in-reply-to<expression>)**
  A performative of this type indicates that the sender cannot understand or considers to be illegal the message referenced by the :in-reply-to parameter.

[UMBC 1993]

Knowledge Interchange Format (KIF) was developed as the content language of the ARPA KSE ACL. A speech-act expressed as a KQML performative takes a KIF as one of its parameters. KIF is a language designed for use in the interchange of knowledge among disparate computer systems (created by different programmers, at different times, in different languages, and so forth).

KIF is a prefix version of first order predicate logic with extensions in order to enhance its expressiveness. The following KIF expression state that chip1 is larger than chip2:

```
(> (* (width chip1) (length chip1)) (*(width chip2)
(length chip2)))
```

[Genesereth and Ketchpel 1994].

[Genesereth 1], [Genesereth and Ketchpel 1994], [UMBC 1999:b]

6.2.3 General and Reusable Ontologies

One aspect of ARPA KSE worth mentioning in relation to the discussion of agents in electronic commerce is the efforts to establish consistent system of ontologies covering different problem domains. Many ontologies have been successfully developed in different domains, but in order to make systems more generic it is desirable that an ontology developed in one domain would also be usable in another domain. But many times concepts are used differently in different domains. In the same way it would be desirable to organize concepts in a hierarchical, tree-like fashion from more abstract to more specified concepts in a way similar to how classes are organized in an inheritance tree in object oriented analysis and design. Both of these undertakings have proved much more difficult than first expected, as described in Chandrasekaran, Russ, MacGregor and Swartout [1999].

6.2.4 Fipa ACL

FIPA – Foundation for Intelligent Physical Agents – was formed in 1996 to support standardization mainly in inter agent communication. An overview of standardization work up till 1997, especially the FIPA 1997 standard, is found in Dickinson [1997]. The work in FIPA, including full specifications, is monitored at the FIPA website, FIPA [1999]. Many links are broken.

6.2.4.1 The FIPA 1997 Standard

The FIPA 1997 Standard consists of two components:

- Normative technology references:
    - Agent Communication Language.
    - Agent Management.
    - Agent/Software Integration.
- Informative reference applications. Dickinson mention four reference applications:
    - Personal Travel Assistance: Using agents to plan a trip, including choosing itineraries, modes of transport, booking tickets.
    - Personal Assistant: Supporting regular activities in the office like diary management, email sorting and workflow.
    - Audio/Video Entertainment and Broadcasting: Selecting from available programs to support customers with hundreds of TV- or other media channels.
    - Network Provisioning and Management: Using agents to provide dynamic virtual private networks (VPNs).

[Dickinson 1997]

FIPA ACL
The FIPA Agent Communication Language, (ACL) defines a message passing language for textual messages. The standard does not define the means for message transport since that is considered platform dependent.

An ACL message can look like this:

```
(inform
   :sender buyer1
   :receiver hpl-auctioneer
   :content
     (price (bid lot07) 150)
   :in-reply-to round4
```

```
      :reply-with bid5
      :language sl
      :ontology hpl-auction
    )
```

[Dickinson 1997]

Inform is a *communicative act type*, and as such define the principal intention of the message – to inform what this agent believes to be true. Communicative acts are based on speech act theory. The FIPA 97 contains twenty communicative act types. The rest of the message consists of seven key-value pairs, most of which are self-explanatory.

FIPA 97 specifies the semantics of ACL messages in Semantic Language – SL, a logic system that includes a model for actions. SL includes operators for belief, intention, having a goal and being uncertain.

FIPA is working on its standard incrementally. Work has been done during 1998 on the FIPA 1998 Standard, v.0.2 of which now is available at FIPA [1999]. Mobility is mentioned in passing in the 1997 version of the standard, but is more elaborately treated in the 1998 version.

6.2.5 Discussion: ACLs, Standards and Complexity

A draft specification of KQML, UMBC [1993] has more than forty performatives. We have not been able to obtain later specifications, but from what we understand the number of performatives has not decreased. The FIPA ACL has twenty communicative act types, the FIPA equivalent to performatives. KQML and KIF was designed to make sharing of large knowledge bases possible and expressive power was required. The four reference applications of FIPA reveal the ambition to achieve a general agents framework applicable to widely different problem domains.

Both of these efforts seem to reveal that building general agent systems involve great complexity. The industry or the research community does not wait for the agent standardization work in FIPA (and elsewhere) to complete but use smaller languages for domain specific applications. De facto standards may or may not evolve in these domains. Eriksson, Finne and Janson [1998:b] take a similar position.

6.3 MAS – Multi Agent Systems

It follows from the definitions of both Genesereth and Ketchpel [1994] and Petrie [1998] that SA-agent systems are multi agent systems. At least one functional and downloadable Java package supporting SA-agent systems is available, namely the JATLite systems of Stanford University, see JATLite [1999].

6.3.1 Load balancing – Market Oriented Programming

SA-agents are well suited for the type of optimization that has come to be called market-oriented programming. The overall model is a market striving for price/supply equilibrium. In this section we discuss some applications. The Agent–based Market Space of Eriksson and Finne is also related to this model.

6.3.1.1 Power Plants and NO_x Emission

Gustavsson [1999] describes how agents can be used in load balancing between power plants. We illustrate the idea with a similar example.

We presuppose a number of power plants producing electricity from fossil fuel. Apart from electricity the plant also emit NO_x in a super linear relation to their power production, i.e. a 10% increase of power results in more than 10% increase of NO_x emission. The relationship between load and emission may also be dependent on the type of fuel currently being used. Load balancing between the plants in order to minimize NO_x emission can be performed in the following way:

- Each plant is represented by an agent.
- The agents can state “I can produce one more kW and thereby emit another M kg of NO_x ”.
- A transaction means changing load from one plant to another.
- A transaction is done if the total emission of NO_x is reduced.
- A call for possible transactions is issued every t seconds.

In this way the system converges to a state where NO_x emission is minimized while adjusting to different load of the entire system. This example is obviously overly simplified. In reality there are difficulties like local minima and deadlock situations, cp. Gustavsson [1999] and Ygge [1998].

6.3.1.2 DiSolver as Load Balancing System

The DiSolver system described in section 5 can be extended into a similar load balancing system. Presuppose a company with a number of desktop computers. The company runs heavy numerical computations and produce reports on word processors running on some of the desktops (the word processors are still managed by human agents). With proper access to the priority cues of the desktops, and provided that the computations are distributable, a design of a multi agent system is possible where the total load of the company is distributed onto the available machines so as to optimize the efficiency of the company.

6.3.2 Glueware

SA-agents are well suited for glueware applications. SA-agent code can be put as adapters in front of legacy code to make an old application available to an agent-based communication system. Cp. Genesereth and Ketchpel [1994].

7 Agents in Electronic Commerce

7.1 Electronic Commerce

The term “electronic commerce”, e-commerce or EC for short, has been established for a few years. It usually refers to business activities carried out with the help of computers in general and computer networks in particular. Talking business on the telephone is normally not considered as electronic commerce, even though telephones are electronic devices.

Specifically, EC refers to the transfer of business documents between computers, systems for integrated stock-keeping, automated ordering, bookkeeping, invoicing etc. as well as to marketing and sales with the aid of computers or computer networks.

Deploying computer networks involves many security issues. Systems for secure transmission of secret data as well as secure payment systems using different cryptological techniques have emerged.

The WWW-explosion with the possibility to market and sell directly to the PC-owning public has boost the interest in EC enormously. On the Internet and WWW there is also the possibility of marketing and selling material affected by IPR* legislation. Thus EC comprises questions for IPR as well as techniques for hindering misuse of copyright material, such as watermarking* of text, images and even computer program code. The possibility to provide systems for “pay as you use” on the Internet have stirred an interest in micropayment systems.

Implementations of EC extend over every possible field of software engineering, from web design to knowledge based systems. There has been a great interest in “agent technology” in the area of EC, especially in the area of automated business activities from advertising, via brokering to negotiating terms of contracts to payment, delivery and post sales services. In the following we look specifically at software agents in the context of a business-to-consumer model. Different kinds of agent technologies have also been used for back office applications like information retrieval.

The Amalthaea info-mining system presented in section 7.4 is one example.

7.2 Business-to-Consumer

In this section we examine some existing web-based solutions for business-to-consumer marketplaces in the perspective of a six-stage business model. Business-to-consumer electronic commerce does not necessarily have to take place on the WWW. But it seems likely that solutions where the web at least is a part will dominate in the foreseeable future. For the public in general the WWW and the Internet are one and the same.

7.2.1 A Business Model

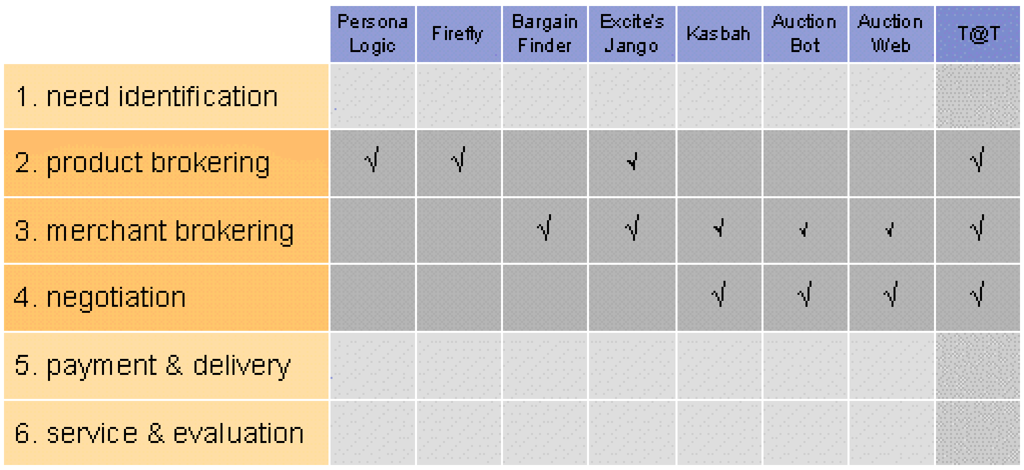
In order to clarify the roles of agents in electronic commerce, we use a business model. We use a slightly modified version of a model sketched out in Maes, Guttman and Moukas, [1999], stemming from consumer buying behavior (CBB) research. This model does not cover all of (electronic) commerce but still serves our purpose here as we concentrate our discussion on the business-to-consumer and consumer-to-consumer domains. The model describes six stages of the business process:

- **Need Identification.** The buyer becomes aware of an unmet need. This stage might include the buyer reading/watching advertisements, talking to friends etc.
- **Product brokering.** The outcome of this stage is a set of products that the buyer seriously considers buying, the consideration set. It might include evaluation of product information from producers, retailers and consumer organizations based on the buyer's own criteria.
- **Merchant brokering.** The outcome of this stage is the decision who to buy from. This combines the consideration set, information from and about retailers and criteria like price, warranty, availability, delivery time and reputation.
- **Negotiation.** This stage considers the terms of the transaction.
- **Purchase and delivery.**
- **Product service and evaluation.** This stage involves post-purchase product service, customer service and evaluation of the overall experience of the product and related services.

These stages are often not distinct in real commerce. The business process often alternates between two or more stages before moving on to the next.

As we shall see, most systems are strongest in product and merchant brokering. Some also negotiate price on behalf of the user. Support for need identification, purchase and delivery or product service and evaluation are normally weaker.

Most of the services discussed in the following are not, as far as we can tell, agent-based in the sense of section 2.5.1.5. They are not based on a speech-act-agent implementation. Some of them are, however, agent-based in the sense that the user experiences a part of the system as performing business actions on his behalf, as his trusted representative, and in some cases that it does so in interaction with other "agents" (in most cases, representing other vendors). Thus they are agents in the sense of MW3 and MW4 as well as agents possibly on the level of modeling and design and on the user-interface level. And as we can see in figure 2, they are alleged to be agent mediated by an authoritative source like Guttman, Moux and Maes, [1998].



	Persona Logic	Firefly	Bargain Finder	Excite's Jango	Kasbah	Auction Bot	Auction Web	T@T
1. need identification								
2. product brokering	√	√		√				√
3. merchant brokering			√	√	√	√	√	√
4. negotiation					√	√	√	√
5. payment & delivery								
6. service & evaluation								

Figure 2.
Consumer buying behavior model
with alleged agent mediation.
[Guttman, Moux and Maes 1998]

Some of the services in the following sections apply to more stages in the business model then the ones under which they are listed.

7.2.2 Need Identification

Some sites use user profiles to custom make each individual user's own version of pages from the site. Amazon.com and their Eyes program is one example. [Guttman, Moux and Maes 1998]. Some commercial sites send e-mail automatically to notify customers of new products in areas in which they have expressed interest.

7.2.2.1 Firefly

Firefly claim that their products make it possible for consumers to specify a user profile to a vendor while keeping his/her identity secret. In an online demo, Firefly states the bookstore Barnes and Noble as an example of a user of their products. When creating an account on the Barnes and Noble home-site, however, we do not get the impression that our preferences on books are in any way withheld from Barnes and Noble. And furthermore, when loading the Barnes and Noble home page after setting our profile, our profile is in no way related to what books is presented on the page. The role of Firefly products on the Barnes and Noble home-site is unclear, as is the agency of the software. [Firefly 1999], [Barnes and Noble 1999].

7.2.3 Product and Merchant Brokering

Quite a few systems support merchant brokering on the basis of price comparison. There has been an interesting struggle going on between BargainFinder-like web-sites and the vendors being displayed by those web-sites. A third of the online merchants accessed by BargainFinder blocked out its price requests. The reason seems to be that they do not want to compete with price alone. More advanced price-comparing software do their searches from the client's web-browsers so that their requests look just like any customers'. To defeat this some merchants have put their price tags in picture-formats like jpeg so that software cannot easily parse the pages to find prices. [Maes, Guttman and Moukas 1999]. That will eventually not stop price comparison, since it is not too hard to train a neural net to map shapes to characters. Text-scanning software is today an off-the-shelf product.

7.2.3.1 Passagen

Swedish Passagen give links to a limited number of retailers. The system does no comparisons of prices. It thus gives a weak support for stages two and three in the model. No agency can be detected more than possibly on the interface level. [Passagen 1999]

7.2.3.2 Personalogic

The company Personalogic provides a web-based decision support system for such diverse “products” as cars, pets, bicycles, holiday trips, cities, collages and politicians. By clicking on links preferences can be stated and eventually a list of one or more options is supplied. Maes, Guttman and Moukas, [1999] list it as an agent system, but we do not discover much autonomy when using the system as it takes a lot of direct manipulation - clicking on links - to get to a final decision. The service possibly merits agency on the interface level. Some indecisive consumers might find it useful though. It took the author of this paper five minutes to find out that his ideal pet would be a hermit crab.

When the product is specified, the system provides links to one or more sellers of the product. Personalogic supports stage two and three in the model. Personalogic has ads on each and every page. [Personalogic 1999], [Maes, Guttman and Moukas 1999].

7.2.3.3 BargainFinder

BargainFinder was an experimental shopping agent for the WWW developed by Andersen Consulting around 1995. It was a system that compared prices in a number of different online CD-stores. The interface towards each vendors web-site was custom made, so it is not agents talking to agents that retrieves the information, but the BargainFinder did work as an agent as it retrieved the information on the behalf of the user, in the sense of MW3 or MW4c and thus an agent on the interface level. Andersen Consulting does not provide the BargainFinder on line, but a search-engine query on “bargainfinder” returns many lookalikes.

Andersen Consulting has a new research project called Pocket BargainFinder intended to work for more than CDs. Documentation is scarce. [Andersen Consulting 1999], [Maes, Guttman and Moukas 1999].

7.2.3.4 Jango

Jango is an advanced version of BargainFinder. See jango.excite, [1999] for one implementation. It can help to find the best offer on a wide variety of consumer products as well as personal contacts, careers, travels and more. Jango solves the “agent-blocking” issue by having their requests originate from their customer’s web-browsers. The interface between Jango and each and every site it searches has in principle to be handcrafted individually even though special software tools can be implemented.

Jango.excite, [1999] can also handle auctions. Any member can put up an item for auction, stating opening and lowest price and date for closing. Any member can place a bid on any auction item.

When time is up the affected parties are notified of the outcome of the auction via e-mail. Purchase and delivery are up to the parties themselves.

This seems fine as long as the parties can meet in the real world and exchange money and goods. But what if you live in different ends of the physical universe? It may sound risky to deliver goods or money to perfect strangers. Excite has a nice way of increasing the transparency of the virtual marketplace. Everyone that places an item for auction is exposed for other members listing comments on the seller. On the auction web pages these comments can be seen next to the name of the seller. Comments usually go along the lines of "Have done business with her before, she kept agreements" or "She did not deliver", "She did not respond on my e-mail".

Jango supports stages two, three and four in the model.

7.2.4 Negotiation

As already mentioned, Jango.excite also handles auctions. In this section we look at some other online auction systems as well as some that support agent enhanced negotiation in other formats.

7.2.4.1 OnSale

OnSale is an online auction house where you can click your way through the site and see what items are for sale. You can enable a "Bid Maker", an agent that bids for you on an item up to your chosen highest bid. OnSale also has some on-site search-tools. OnSale has ads on every page. Most sellers on OnSale are established retailers. There are also sales as fixed prices on the site. OnSale supports stages two, three and four of the model. [OnSale 1999]

7.2.4.2 eBay

eBay is an online auction house that focuses on the consumer-to-consumer market. In eBay you also have on-site search tools to help you find the item of your choice. You instruct your eBay "proxy" (not agent!) about your highest bid, and the proxy handles your bidding for you. The parties handle payment and exchange of goods themselves. You can read the feedback other buyers have given to a registered seller. The feedback can be positive or negative and is summed up in a feedback index. If it drops under minus four the sellers' account is closed automatically.

eBay has no ads on their pages. Viewing, bidding, and buying items on eBay is free, while listing and selling items on eBay cost money. eBay also supports stages two through four of our business model. [eBay 1999].

7.2.4.3 AuctionBot of University of Michigan

AuctionBot is a research site at University of Michigan. It was in bad shape when we tested it in March 1999. [AuctionBot 1999]

7.2.4.4 Bid2Day

The Swedish online auction house is very similar to OnSale. You click your way to the product and place your bid. The system has no support for agents in the bidding process, i.e. you have to monitor the auction of your choice yourself and place a higher bid if necessary. For some products they do support purchase and delivery. [Bid2Day 1999].

7.2.4.5 Amazon

Amazon launched an auction system similar to eBay in late march 1999. [Amazon 1999]

7.2.4.6 Kasbah and Market Maker

The Kasbah project of MIT media lab was an early example of an agent-based consumer-to-consumer transaction system. It has now been replaced by the Market Maker Testing Site. On Market Maker you can browse the site to find interesting goods. The categories available are limited to Books, Music, Star Ships, Computer Games and Translation Services. In each of these categories you can specify values in a number of fields. For books these fields are Cover Type, Genre, Title, Author(s), Edition, Course Number, Condition and a free text Description. In the web interface you can create agents that buy or sell items according to your specifications. You state the life span of your agent and its negotiation strategy by choosing from three different time/price functions, cf. figure 3. If you sell you let your agent start out on a high price and decrease it to a lowest acceptable price according to the chosen time/price function. Corresponding rising functions are available if you buy.

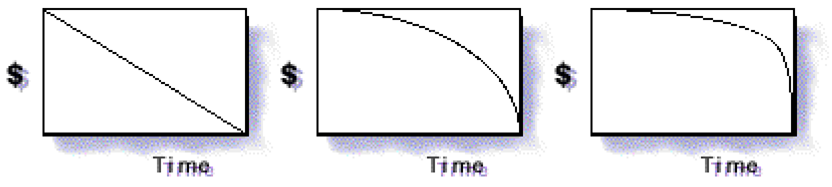


Figure 3.
Different pricing strategies for the Market Maker selling agents.
(Source: MIT Media Lab)

The system supports stages two through four in the business model. Market Maker has a feedback system called Better Business Bureau similar to the feedback systems in OnSale and eBay, giving support for stage six in the model.

We have as yet been unable to obtain information on the Market Maker implementation. It seems probable that it is based on speech-act agents.

The negotiation method of deciding on prices is a nice demonstration of the power of agents. It can be questioned, however, if it is a sound way of doing business. We suggest that the auction format is a better one for the seller. [Market Maker 1999], [MIT Media Lab, Agents 1999]

7.2.4.7 Tete-à-Tete

The latest (?) agent enhanced marketplace experiment from the MIT Media Lab is called Tete-à-Tete. Contrary to most other transaction systems Tete-à-Tete does not only take price into consideration, but the seller and buying agents negotiate over multiple terms of a transaction, including warranties, delivery times, service contracts, return polices, loan options and gift services. The participants state the relative importance of the different aspects of a deal. The system uses XML to describe the different proposals, critiques and counterproposals. No online demos are available. [Tete-à-Tete 1999]

7.2.5 Purchase and delivery

Normally purchase and delivery is not automated. But it is very easy to click on a link on a web-site to confirm that you buy what is in your chart like you do for example at Amazon.com.

7.2.5.1 Autogiro

In the midst of the WWW-hype it is nice to point out that most banks have provided an automatic payment system in many years now. In Sweden this service is called Autogiro. You can instruct your bank to transfer a fixed amount to another account on a fixed date every month.

7.2.6 Product service and evaluation

The feedback given on sellers in OnSale, eBay and Market Maker is an evaluation of the transaction. Most commercial web-sites are anxious to get feedback from their users on the quality of their services.

7.3 A general open Agent-based market

The open worldwide agent-based marketplace for business-to-business, business-to-consumer and consumer-to-consumer transactions has been a vision for some time. What would such a marketplace look like? How would it work?

7.3.1 SICS Market Space

Joakim Eriksson, Niklas Finne and Sverker Janson at SICS has developed one version of the open agent-based marketplace. The following presentation is based on Eriksson and Finne [1997] and Eriksson, Finne and Janson [1998:b]. We call their framework the SICS MarketSpace, the term used in [1998:b]. It is based on speech-act agents with an outer language, MIL, expressing how you make business in the system and an inner content-language, MIF, describing what you make business with. To make this clearer, let us schematically describe a transaction in the system. In this example there are three agents: a buyer, a merchant broker and a seller. The merchant broker plays the role of the yellow pages in the telephone directory. This example does not follow the syntax of MIL and MIF but serves as an illustration of a possible transaction.

A: Buyer	**B: Merchant Bro-ker**	**C: Seller**
Brokering		
1. A to B: (*Tell (*A *buy* X))		
	2. B to {C, F, H, J, K}: (*Tell* (*A buy* X))	
Negotiation		
		3. C to A: (*Offer* (C *sell* X *Price* p) *MessageID* m1)
4. A to C: (*Regarding* m1 *Offer* (A *buy* X *Price* p – 10) *MessageID* m2)		
		5. C to A: (*Regarding* m2 *Offer* (C *sell* X *Price* p – 5) *MessageID* m3)
6. A to C: (*Regarding* m3 *Accept* (A *buy* X *Price* p – 5) *Message ID* m4)		
		7. C to A: (*Regarding* m4 *Accept* (C sell X *Price* p – 5) *Message ID* m5)
Deal is closed		

Brokers can keep lists of other brokers and register themselves as brokers with other brokers. In that way you can get a distributed network of brokers that buying and selling agents can utilize for business.

7.3.1.1 Information Model and MIF – The Market Interest Format

The information model of SICS MarketSpace is based on the notion that participants in the market want to close deals. The basic unit of information is *contract*, a structured document. In order to identify possible deals participants exchange *interests* that represent sets of contracts. Examples of interests that can be expressed in sets of contracts are

- Buy thing cheaper than one dollar
- Buy pizza within one hour
- Buy books on software agents

Contracts are defined in terms of *concepts* (the record types that are the building blocks of structured documents) which in their turn are defined in *ontologies* (collections/modules of concept definitions). Concepts are defined by URLs. Thus a concept identifier http://somesite.dom/basic.ont#contract-3 refers to a definition of “contract-3” in ontology http://somesite.dom/basic.ont. Concepts serve as building blocks of structured documents. No semantic information is attached to a definition other than the types of its components. They are similar to records in programming languages.

An *Expression Of Interest* (EOI) is a representation of an interest in some language. KIF (Cf. section 6.2.2) or the FIPA ACL SL could have been possible candidates, but in their general form these languages are too complex to be convenient for this application and useful subsets of them are not defined. It is also possible that W3C RDF/XML will be possible to use [W3C 1999]. At the time of its writing the SICS MarketSpace proposal [1998] use a simple custom designed language, the Market Interest Format, MIF, to encode EOIs. MIF has Lisp-like syntax. Figure 4 show an example of a MIF definition and a MIF expression:

```
(def car “trade-object”
    (color (instance “pathone-color”))
          …)

(instance “contract-3”
    (date (interval 1/1/98 6/30/98))
    (buyer (instance “person”
                       (name “Joe Smith”)
```

```
        (agent-address…)))
    (goods (instance "car"
                (color (instance "red")))))
```

Figure 4.
A MIF definition and expression

The EOI in figure 4 states that Joe Smith is the buyer of a red car in a contract signed between 1/1/98 and 6/30/98. This could be used by Joe to advertise an interest or by his agent to handle incoming queries.

Values of the fields in EOIs can be expressed as fixed values as well as intervals or lists as to make possible EOIs like “I want to buy a red car for 200 – 300 dollars” or “I want to buy a Volvo 745 from 1998 or a SAAB 9000 from 1997”. As ontologies are hierarchically defined EOIs like “I want to buy Kitchen ware” can render responses containing both dishwashers and refrigerators.

7.3.1.2 MIL – The Market Interaction Language

MIL is the outer language of the system. It is small but powerful enough to implement a variety of different business interactions, like

- Advertisement – promoting your own interest.
- Searching/Querying – to find matching interests.
- Negotiating – to come closer to a deal.
- Offering and accepting a deal settlement.

Figures 5 and 6 show the performatives in MIL and an example message.

Message	Meaning
Non-committing	**Following messages are not legally binding**
Ask(A, B, EOI)	Tell me an interest that matches EOI
Tell(A, B, EOI)	This EOI is an interest
Negotiate(A, B, EOI)	Give me an offer that matches EOI
Committing	**Following messages are legally binding**
Offer(A, B, EOI)	This signed EOI is an offer
Accept(A, B)	This is an accepted ("positively" countersigned) offer
Decline(A, B)	This is a declined ("negatively" countersigned) offer
Meta-messages	**Following messages are about earlier messages**
Error(A, B)	A informs B that A did not understand last message from B

Figure 5.
Performatives of MIL – Market Interaction Language

```
(offer
    :from "map://onesite.dom/agent1
    :to "map://othersite.dom/agent2"
    :in-reply-to i
    :reply-with j
    :language "MIF 1.0"
    :content "<MIF expression>"
            )
```

Figure 6.
An example MIL message.

The in-reply-to and reply-with fields contain simple identifiers to enable agents to distinguish one chain of conversation from another, cp. the "regarding" in the previous example.

We include another interaction example to show the flexibility of the proposal.

- **Ask (A, D, "Sell me a refrigerator)**
  A asks the directory service D for interests matching an interest where A is the buyer and the good of type refrigerator.

- **Tell (D, A, “B and C”)**
  The directory service replies with an interest where A is the buyer and B and C are possible vendors.
- **Negotiate (A, B, “Sell me a refrigerator”)**
  A asks B for an offer where A is the buyer of a refrigerator.
- **Negotiate (A, C, “Sell me a refrigerator”)**
  A asks the same of C.
- **Offer (B, A, “Electrolux 3117B for $350)**
  B gives A an offer (a signed interest) where B is the seller and A is the buyer of a specific refrigerator at a price of $350.
- **Offer (C, A, “Electrolux 3117B for $300)**
  C gives A the same offer but for the price of $300.
- **Offer (A, B, “C sells for $300”)**
  A prefers B as a vendor and forwards C’s offer to A to B, suggesting to B to match or improve on the offer.
- **Offer (B, A, “Electrolux 3117B for $300)**
  B gives A an improved offer.
- **Accept (A, B, “B’s last offer”)**
  A accepts B’s offer.
- **Decline (A, C, “C’s offer”)**
  A declines C’s offer.

This transaction auction-like characteristics as A offer B to improve on the offer. The authors of the proposal believe that a wide range of auction types can be supported by MIL and MIF. The SICS MarketSpace is symmetric, any participant can play the role of seller, buyer, merchant broker etc.

7.3.1.3 Web Integration

SICS MarketSpace is integrated with the WWW. Figure 7 illustrates how a user gets access to his agent as it contacts a service agent. The user has his ordinary web browser plus another smaller window that holds the agent user interface. When the user encounters agent augmented services through the web the user agent is informed and can be given the opportunity to assist the user. The agents communicate with the Market Agent Protocol (map) which is plain TCP/IP transmission of text messages.

[Eriksson and Finne 1997], [Eriksson, Finne and Janson 1998:b].

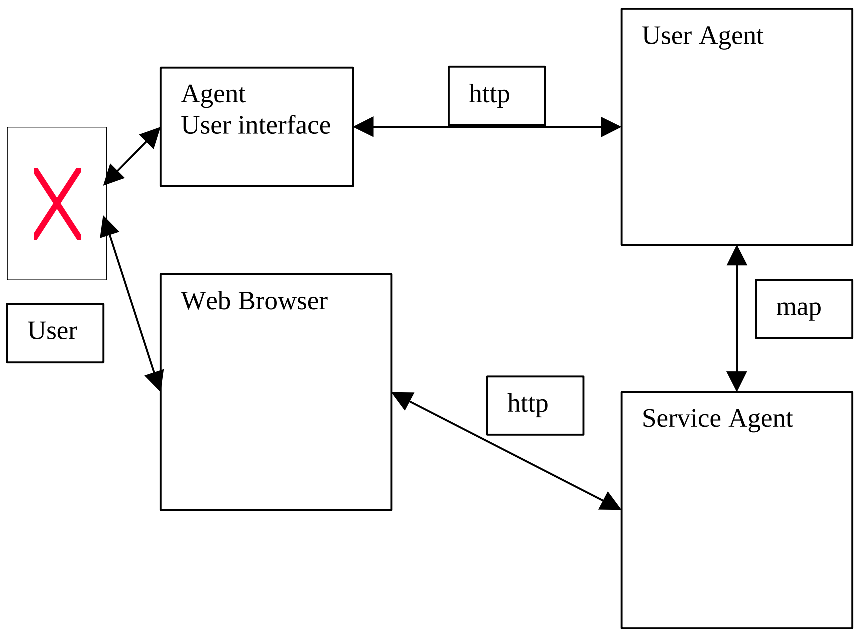


Figure 7.
SICS MarketSpace web integration

7.3.2 Mobile or Stationary?

There is an ongoing discussion whether an automated EC trading system should be based on mobile or stationary agents. As we have already pointed out, mobility and speech-act agency are independent properties. A mobile agent can reside on the end users private PCs and thus be totally controlled by him or her. But most people are not interested in administrating agents per se, they are interested in the services that an agent-based market can provide. It seems likely that this market, if ever, will be brought to the public as a service by an "agent service provider". The business logic of the open agent-based marketplace is discussed in section 7.6.

7.4 Info Mining – Amalthaea

Information mining refers to techniques for assisting in finding information a user might be interested in given what he or other users similar to him has been interested in earlier.

In the Amalthaea project of MIT Media Lab the Artificial Life metaphor has been used to create an information-mining system for the WWW. The basic "live" agents of the system are vectors of weighted keywords called Information Discovery Agents, IDs. WWW-documents are internally represented by an ID where the weight of each keyword represents the frequency of the word in the document. Common words like *the, and, it* etc. are omitted. Thus, web pages, or properties of web pages, are mapped onto $\mathbf{R}^n*$.

The user can also state an interest as an ID. A (logical) population of such IDs is represented internally. The system can then look for WWW documents with the same or similar ID. The similarity between IDs can be represented by the distance between their two points in $\mathbf{R}^n$. Small distance means big similarity.

Adaptive mechanisms are built into the system in the following ways (simplified):

- The user gives feedback to the suggestions of the system. An ID that participates in the delivery of a suggestion approved of by the user is rewarded with credits. If the suggestion is not approved of, the ID looses credits.
- The IDs do not suggest a document if it is not sufficiently convinced that the suggestion will be approved of by the user.
- Some IDs "mutate" and "mate". When they mutate the keyword weights are changed randomly within certain constraints. When a couple of IDs mate they exchange a part of themselves, i.e. a section of the vector with keywords and weights.
- IDs multiply regularly according to the amount of credits they have previously collected.
- When the overall level of ID-credits in the system goes down evolution is boosted by rising the frequency of mutation and mating. When the credit-level is high mutation and mating is low, i.e. the system is stable.

If the user searches for documents in different subjects the system can adapt by entertaining different populations of similar IDs. If an interest of the user changes over time, the ID-population of that interest changes their weights and keywords in an evolutionary way. Successful IDs prevail and others perish. No commercial companies have capitalized on the results of the Amalthaea project or similar ideas, even if many companies have worked on code from the project. [Moukas 1996].

7.4.1 Design Metaphor

Amalthaea seems to be an example of using the agent metaphor at the design level. The model of the design is based on that a part of the system behaves logically as a population of independent autonomous “live” agents that act in their own interest in multiplying. That does not necessarily imply that every “individual agent” needs to be implemented as an object by itself. It would be more effective in this case to keep a list of the different “specie” and append an integer to each of them representing the number of individuals.

7.5 Legal Aspects

By Heléne Wallgren

The deployment of software-agent technology raises some interesting legal issues. We will discuss questions concerning the legal identity of agents, their legal capacity and issues concerning accountability, evidence and privacy. Our objective is not to present an exhaustive discussion on the legal aspects of agent technology, nor to give answers to the legal questions that we raise.

Current technology makes it possible for software agents to make agreements. One area where autonomous agents can be used is in electronic commerce. A user can instruct a software agent to investigate prices for one or several products. The agent can be instructed to negotiate the price and, if the parties agree, come to an agreement on purchase. To illustrate what kind of legal issues that may arise we use an example.

> John buys an agent program at Computers and Stuff Inc. He installs it at home and decides to test it. John has heard that agents can be used for online auctions and to buy and sell goods on line. He decides he wants to go to Malta. He gives his agent instructions to find the cheapest trip to Malta. He does not want to pay more than SEK 3500 for a week’s vacation. John is not a skillful operator of computer software and does not understand that he instructs the agent to look only for trips to Malta for exactly SEK 3500, no more, no less. The agent finds a travel agency, SeaSide Travel, that provides a trip to Malta for exactly 3500 and John’s agent and the travel agency software close an agreement. John receives his ticket by mail. A few days later he discovers an ad in the newspaper. SeaSide Travel is now selling trips to Malta for SEK 2500. John calls SeaSide and finds out that he could have

bought his trip at this lower price. But John has now paid and SeaSide is unwilling give him a refund.

This example illustrates some legal issues related to agent technology. The first question is whether the agent can make an agreement that is legally binding for John. The next question is who is responsible for John's misunderstanding about how to operate the agent program. Should the retailer Computer and Stuff be obliged to inform John about how to operate the software? It could be the case that the program interprets John's instructions erroneously. If so, who is then responsible, John or the manufacturer of the software? When John discovers that he could have saved SEK 1000, he may want to claim there never was an agreement between him and SeaSide. Who then bears the burden of proof?

This leads to the question of what legal status should be ascribed to software agents. A legally binding agreement requires that the parties are legal entities. As yet only natural and juridical persons are legal entities. Furthermore the legal entities have to have legal capacity. This means that a natural person has to be of age and not be legally incapacitated. Legal representatives of juridical persons have legal capacity. This seems to imply that it is unlikely that a software agent could be regarded as a legal entity with legal capacity to exercise rights and assume liabilities. It would lead to complicated disputes in instances of disagreement.

Which legal status should be assigned to software agents? One way would be to rank use of software agents for making binding agreements with warrant constructs. A software agent could be looked upon as a technical middleman corresponding to an authorized attorney. In this case the user of the agent would not be responsible for malfunction of the agent. This view is in conflict with what we just said about software agents lacking the status of legal entities with legal capacity able to exercise rights and to assume liabilities. [Hultmark 1998].

Another way is to look upon software agents as immediate dispositions of their end users, i.e. the persons instructing the agents. A software agent is a representative of someone who has given it instructions to obtain certain information or to do business in a certain way, but that does not make the agent any more responsible than any other tool used by man. It would indeed lead to bizarre situations if software agents were to be looked upon as legal entities. Can we put a software agent in jail if it commits a crime? Can a software agent be obliged to pay for damages resulting from its actions?

We have reached the conclusion that software agents should not be considered legal entities. It then follows that they cannot be held responsible for their actions. Who then is responsible? We can consider four possibilities: 1) the creator of the software, 2) the retailer of the software, 3) the one who provides the software for use and 4) the end user of the software, i.e. the one who instructs the agent. Often software manufacturers renounce any responsibility for consequences resulting from use of their products. This is not consistent with Swedish consumer-rights legislation as this legal framework makes such renunciation illegal. Our purpose here is not to solve the question of responsibility but to point out some legal issues that remain to be solved.

If a dispute should occur between parties about an agreement closed with the help of software agents regarding for example a purchase, the parties need to be able to present proof that an agreement has been closed. Swedish courts practice free submission of evidence, which means that in principle any type of evidence is admissible in a trial. This implies that it should be possible to refer to any data as proof that a specific event with legal implications has taken place. This infers that it is important to be able to demonstrate that agent systems have a high level of security. Security systems might include logging, access control and encryption.

Agents do not only perform business. Some agents are designed to “learn” a user’s behavior by registering what kind of information or service he requests or what kind of goods he buys. The data that thus occurs need to be protected. It is in some cases possible to establish the identity of a physical person from for example a user ID or an email address. This raises the question of how to preserve the individual’s privacy and integrity.

From the discussion above we can conclude that the jurisprudence regarding software agents is at least unclear if not incomplete. In order to obtain clarity regarding legal aspects of agent technology we should await legislation or precedential judgements.

(Translated from Swedish by Jonas Edlund)

7.6 Summary and Discussion

7.6.1 Business Logic

While agents in electronic commerce have been a hot topic for some time now, we have not yet seen the general shopping agent, roaming the entire Internet in order to find the best buying opportunity for its master. Whether the agents of SICS MarketSpace or anything like them will fill that niche remains to be seen. The foundations for secure and effective management of migration, messaging, including forwarding of messages, storage and restorage of mobile agents seems to be at hand in commercial systems. We also have the knowledge and technology to implement an effective inter-agent communication language. Secure economic transactions over the net also seem to be a working reality. Maybe the business-to-business market (B2B) will be the first to enjoy the new technologies. In U.S.A., the information-consulting firm Forrester Research in Cambridge, Mass., has estimated the B2B market to be worth about U.S.$183 billion by 2001, while the business-to-consumer (B2C) market is only estimated to U.S.$17 billion at the same time. [CACM 1999].

The reluctance of established Internet marketplaces to lay themselves open for agent inspection of prices etc. has hampered the emergence of general consumer shopping agents. This is understandable. OnSale have advertisements on every page so selling ads is a substantial part of OnSale's income. Software agents are probably not open for influence from ads.

We note that marketplaces like OnSale and eBay do not need to use general shopping agents in their systems implementation. Maybe the trend towards "meta-shops" like excite will continue towards "meta-meta-shops" covering each others supply of offers. That could possibly result in loops in the overall system resulting in confusing duplicate offers a problem that could possibly pave the way for an open agent-based market in the line of the SICS proposal.

What business logic conditions are at hand for an open agent-based marketplace? The SICS MarketSpace is developed in close collaboration with Telia Research, an R&D branch of the biggest Swedish Internet Service Provider. How can, or will it, be brought to market? Telia can put up their own agent facility and provide agents from a web site. If they do, will they restrict agent behavior to enhance their own interests? That seems unlikely. Initially an agent provider would probably give the system away for free as to establish a critical mass of usage in some specific sectors of retail with easily defined products, like books or CDs. Once the system is established the agent provider can charge the users of the agents just like a telephone company charges for using the telephone. The business logic of open agent-systems ought to be similar to that of telephones; the provider gains more the more we use it.

An agent provider could further extend an open agent system with a jango.excite-like meta-facility to access vendors with no agent enhancement as to make their offers accessible to its agents. This wold add some complexity to the design of the agent system as deals on these offers cannot be closed by the agents, but this should not be a big problem. The agent could deliver the results of the overall query to the user by email. In such a way the open agent-based marketplace could benefit from the offers made by the meta-shops and possibly hamper the turnover of those meta-shops that make money from ads.

If we look at vendors it is likely that small vendors would be happy to deploy open agent-systems as a means of boosting business, while bigger vendors with already established web-shops would be more reluctant. But as an open system starts to spread, they might open up for agents, by self-interest, at least if they do not loose advertising potential in the process, as sketched above. Thus a shop like eBay that does not provide banner-space for others but charge the sellers would be in a stronger position in this regard than would jango.excite or OnSale. At any case, web-shops can keep their old web-based systems up while connecting to the “agent web”.

An agent EC system can be designed so that users get better control of how they give other parties access to their personal profile. A user profile is valuable for a vendor in a web encounter as the vendor can customize the presented page to present relevant offers. The buying agent can choose to withhold its specific interest at a query stage by presenting more general interests to the potential seller and filtering the information before presenting it to the buyer. Giving away your profile could possibly result in lower prices.

The need for repression mechanisms towards malignant agents was briefly touched upon i section 6.1.

7.6.2 Metadata Descriptions and Interoperability

One subject closely related to software agents in EC is the question of metadata descriptions of content on the web. The W3C RDF/XML was mentioned in the section on SICS MarketSpace. Similar work is also carried out in the CommerceNet consortium especially in their eCo project. The aim of CommerceNet is to make interoperability possible between major systems for EC such as Open Trading Protocol, EDI, RosettaNet, Open Buying on the Internet (OBI), Information and Content Exchange (CNET) and open Financial Exchange (OFX). [Glushko, Tenebaum and Meltzer 1999], [CommerceNet 1999:b].

7.6.3 Future Winners

Who will come out as winners in the battle of systems and standards for the different areas of electronic commerce? Or more cynically: Who will be able to impose their solution on others? We conclude this section on EC by looking at some categories of players and their potential to succeed and impediments they may encounter. This may be important for computer scientists that want to get agreement on a good technical solution. The outcome of the competition between different technologies is not dependent only on the quality of the technology, but also on the incentive-patterns of the players involved, as well as, of course, on politics and money.

7.6.3.1 Big System Providers – CommerceNet

Big system providers can often impose their solutions on the market, especially if they team up. The CommerceNet consortium and its according to CommerceNet [1999:a] 186 members in March 1999, including IBM, Microsoft, Sun, Netscape, the U.S. General Services Administration, American Express, Cisco, Ericsson, GM and HP, have invested vast resources in establishing workable standards in e-commerce, especially in the business-to-business domain.

Potential

Once the members of CommerceNet agree, they can point out the way to the future with a very big hand. New e-commerce players on the verge of equipping themselves with new systems will look very closely to CommerceNet's proposals, either directly or as presented to them by vigilant e-commerce consultants.

Impediments

Big organizations are slow. It takes time to agree on new solutions. And the old players in e-commerce that already have systems that function well (c.f. next section) may be reluctant to assimilate new ways of doing business. If it works, don't fix it!

7.6.3.2 Multinational Non-software Companies

Big companies can impose their own system on their own supply chain. Cf. Gustavsson [1999].

Potential

A high level of optimization in the supply chain can be achieved with a multi agent system. If management sees the advantages they will possibly get the implementations and the big company will in that way help spreading good technology.

Impediments

Big companies can be slow on issues outside their own product line.

7.6.3.3 Research Labs of Universities and Independent Institutes

Much of the new technology in e-commerce has been created at universities or independent institutes.

Potential
High level of competence, a milieu where creativity thrive, and some freedom to set up promising projects.

Impediments
Why take the risk of starting a business when you have a steady paycheck from the university? “Someone else can build the final system, I do research!”

7.6.3.4 Small Companies with Win-Win Solutions

A small company with a win-win solution can change the landscape of the electronic marketplace rapidly. The history of electronic commerce is full of examples: Netscape, Yahoo, Excite (or why not Microsoft…?).

Potential
The small company can make decisions fast and act swiftly. A small company can be formed on a single effective business-solution with no hampering considerations to legacy issues, by a small team of dedicated people.

Impediments
A small, unknown company may have problems making its voice heard. But then again, the Internet and WWW present enormous potential for the innovative newcomer.

A small company may lack the critical mass of competence to realize a solution not thought of by the big players. But then again, the right mix of competence can often occur when the right people meet within the framework of a research institution, a big company or a conference.

8 Impersonating Agents

Impersonating agents are digital or computerized actors. They convey a believable persona to the spectator/user. In 1966 Joseph Weizenbaum at MIT wrote a Fortran program called Eliza based on string matching, impersonating a psychologist via a text-based user interface. Rumor has it that his secretary found it so believable that she asked the program for advise on whether or to leave her boyfriend. Weizenbaum, [1966] is a proper source for Eliza that we have not yet obtained. Modern Elizas are available on the web, e.g. Josef Stefan Institute, [1999] and Goerlich, [1997].

There is an increasing interest in what can actually be done with computers in the area of creating believable characters. Modern techniques like voice synthesis, voice recognition and 3D-graphics make it possible to turn a computer into a virtual theater.

Impersonating agents qualify by definition to agency at the level of interface. Some of them seem to do nothing more, but others are based on techniques that give more autonomy to the object that represents the agent in the computation, as does Creatures.

8.1 Interface Level

8.1.1 Virtual Friend

The California-based company Haptek has created 3D characters called Virtual Friend that communicate with voice synthesis for output. As yet, they use text messages for input. It is not very intelligent, as it is only capable of holding monologues or returning the input text as speech, but that it does very well. It can also sing a few songs and the characters come across in nice 3D graphics. Downloads are available for free at Haptek [1999].

8.1.2 Sylvie and Julia

Sylvie of Virtual Personalities is a chatterbot, a chattering robot. In the downloadable demo she interfaces the user through a rather static graphical interface displaying her face and moving her lips as she speaks. She requires text for input and answers with speech synthesis just like Virtual Friend but she also displays a superficial intelligence. Sylvie is a descendant of the famous Julia, a MUD* interfaced chatterbot created by Michael Mauldin, co-founder of Virtual personalities. Like Eliza, Julia was based on string-matching technique and also grouping of conversation topics. She is presented at length in Foner [1993], a paper available at online at MIT media Lab, and receiving more hits than any other of their papers.

We must unfortunately issue a warning regarding the Sylvie 1.36 demo download. *Do not ask Sylvie for the value of (pi)!* We did, and the bitch stalled the machine, a PC running Windows NT 4. Cp. Foner [1993]. [Virtual Personalities 1999]

8.2 Implementation level

8.2.1 Artificial Life: Creatures

The British company CyberLife has created an entertaining product line called Creatures. The software displays characters in flat but layered graphics (2½D). The agents impersonating these characters are implemented as autonomous objects in a bottom-up fashion with their own genes and metabolism and a neural-net “brain” for self-modifying response to stimuli. The characters live in a virtual world where they play with their toys, search for food and also communicate in a simple verb-object language. They eat, grow, mate and reproduce. When Stephen Grand, chief technician of CyberLife presented Creatures at the Autonomous Agents ’97 Conference the audience listened with growing suspicion, until he run the software. It actually worked as he had described. Grand and his company are currently looking at other application areas for their technology. [Creatures 1999], [CyberLife 1999:a], [CyberLife 1999:b], [Elliot and Brzezinski 1998].

8.2.2 Real Life on Computing Substrate

In his IEEE Expert article, Grand [1997], the author argues for the possibility of real life on a computing substrate. If, he argues, we can represent elementary particles in computers reasonably accurately, then we can also represent atoms, and then molecules, proteins, cells and ultimately living creatures. Thus we could create a virtual world inhabited by creatures that actually are alive, not just look as if they were. We can argue that those creatures would live on the substrate of the computing machinery in the same way as we live on the substrate of self-organizing matter. We can even argue that this is what Grand and his colleagues have done with Creatures, even though they have included some top-down constructs in order to create them and keep them in place.

We want to make clear what we mean when we say that humans live on a substrate of self-organizing matter and energy. According to contemporary physics, at the Big Bang the universe was made up of nothing but self-organizing matter and energy, and, as far as physics is concerned, it still is. Mainstream modern physics have no theories on spirit as something qualitatively different from energy and matter and that is an autonomous agent in its own, i.e. physics do not believe in spirits. In biology the situation is the same and physicians of today do not expect to find the part in the human body where the actual "person" resides, like did their colleagues in the seventeenth and eighteenth centuries. Looked upon as an object of physics, the universe still contains nothing but self-organizing matter and energy. So that is what rocks are and that is what humans are composed from. We live and breed on a substrate of self-organizing matter and energy. In a way the distinction between matter and spirit turns into a question of abstraction level. We perceive each other as humans, having a human gestalt*. We *exist* at that level, we are *composed* of self-organizing matter and energy.

For more on artificial characters, see Extempo [1999] or Maes [1995:b].

9 Conclusions

As software becomes more complex and more tasks are automated and taken over by computer systems two issues emerge of interest in regard to software agents:

- We prefer complex systems to be autonomous. This means adapting to changing environments and ability to handle complex tasks without direct manipulation from users. Different agent technologies, some of them presented in this paper, are, and will increasingly be of great help in achieving this goal.
- We want software to support us and to be user friendly. Agency at the level of user interface means software that help us in our daily lives, on- and offline.

There are strong implications that open agent-based systems for automated electronic commerce will be deployed to the benefit of consumers and small companies. A transparent web accessible market can connect consumers and producers on a global basis. This will have far-reaching consequences for business as we know it today.

No doubt, concepts like "autonomy", "intelligence" and "agency", when applied to software, are seductive to the way we think. If you are not a sales person, the less you anthropomorphize your software, the better off you are. But then again, every once in a while, we are all salespeople.

10 References

Enclosed links were not broken at the time of writing, November 1998 – April 1999.

Agent Society (1997:a). Report on First Meeting of the Agent Interop Group 9 – 10 Feb 1997. http://www.agent.org/society/meetings/workshop9702/report.html.

Agent Society (1997:b) Open Interoperability Protocol and Frameworks, Design Activities. http://www.agent.org/pub/satp/satp-index.html.

Agent Society (1998). Home site. http://www.agent.org/.

Agha, G. (1986). ACTORS: A Model of Concurrent Computation in Distributed Systems. The MIT Press: Cambridge, MA.

Agha, G., Wegner, P., and Yonezawa, A., editors (1993). Research Directions in Concurrent Object-Oriented Programming. The MIT Press: Cambridge, MA.

Amazon, (1999). http://auctions.amazon.com/.

Andersen Consulting, (1999). http://www.ac.com/services/cstar/cstar_child/ecpocketbf_cn.htm.

AuctionBot, (1999). http://auction.eecs.umich.edu/.

Austin, J., L. (1962). How to Do Things with Words. Clarendon Press.

Bates, J. (1994). The role of emotion in believable agents. Communications of the ACM, 37(7):122 – 125.

Banchs, A. (1997). Related Links. On line list of mobile code projects. http://www.icsi.berkeley.edu/~banchs/work/links.html.

Banyan, (1999). http://www.banyan.com.

Barnes and Noble, (1999). www.barnesandnoble.com/.

Bates, J., Bryan Loyall, A., and Scott Reilly, W. (1992a). An architecture for action, emotion, and social behavior. Technical Report CMU-CS-92-144, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

Bates, J., Bryan Loyall, A., and Scott Reilly, W. (1992b). Integrating reactivity, goals, and emotion in a broad agent. Technical Report CMU-CS-92-142, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

Bid2Day, (1999): http://www.bid2day.com/.

CACM (1999). Communications of the ACM, March 1999/Vol. 42, No. 3.

Chandrasekaran, B., Russ, J., MacGregor, R., Swartout, B. (1999). What Are Ontologies, and Why Do We Need Them. IEEE Intelligent Systems, vol. 14, No. 1, 1999. January/February 1999.

Cheng, Y. (1997). A Comprehensive Security Infrastructure for Mobile Agents. Department of Computer and Systems Sciences, University of Stockholm/Royal Institute of Technology.

Chess, D. M., Harrison, C. G. and Kershenbaum, A. (1995). Mobile Agents: Are they a good idea? Research report, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598. http://www.research.ibm.com/massdist/mobag.ps.

CommerceNet (1999:a) Web-site, members list. March 1999. http://www.commerce.net/cgi-bin/memshow/memalpha.cgi?alpha=all.

CommerceNet (1999:b) Web-site. http://www.commercenet.com/

Creatures (1999). Home-page http://www.creatures.co.uk/.

CyberLife (1999). http://WWW.cyberlife.co.uk/.

CyberLife (1999:b). What is CyberLife? http://www.cyberlife.co.uk/cyberlife/archives/whatiscyberlife.html.

Dickinson (1997). Agent Standards. HP Technical Report. HPL-97-156. Hewlett-Packard, HP Labs Bristol Agent Technology Group. http://agents.hpl.hp.com/papers.htm.

Dartmouth (1997). Overview of mobile Agents. http://www.cs.dartmouth.edu/~agent/general/overview.html. Last updated on January 27, 1996.

Dartmouth (1998). D'Agents. http://www.cs.dartmouth.edu/~agent/

Dawkins, R. (1976). The Selfish Gene. Second edition 1989. Oxford University Press. Paperback (October 1989) ISBN: 0192860925.

eBay (1999). http://www.ebay.com/.

Eisenhardt, K. (1989). Agency Theory: An Assessment and Review. Academy of Management Review, pp. 57-74.

Electronic Commerce (1997). http://www.t-stone.co.uk/ec/glossary.html.

Elliot, C., Brzezinski, J. (1998). Autonomous Agents as Synthetic Characters. AI Magazine vol. 19, no. 2. AAAI.

Eriksson, J., Finne, N. (1996). MarketSpace: An open Agent-based Market Infrastructure. Computer Science Department, Uppsala University. Masters Thesis. http://www.sics.se/~joakime/thesis.ps.

Eriksson, J., Finne, N., Janson, S. (1998:a). To Each and Everyone an Agent: Augmenting Web-based Commerce with Agents. Intelligent Systems Laboratory, Swedish Institute of Computer Science.

Eriksson, J., Finne, N., Janson, S. (1998:b). SICS MarketSpace: an agent-based market infrastructure. The ISL laboratory at SICS. Proceedings of the 1998 Workshop on Agent-Mediated Electronic Trading. Springer-Verlag, 1998. http://www.sics.se/publications/marketspace.html.

Ethington, R (1998). Distributed Object Computing. Feature story in Java Report Online.

Etzioni, O., Lesh, N., and Segal, R. (1994). Building softbots for UNIX. In Etzioni, O., editor, Software Agents – Papers from the 1994 Spring Symposium (Technical Report SS-94-03), pp 9 – 16. AAAI Press.

EUNett (1999). http://nettvik.no/pressensplass/agenter/.

Extempo (1999). http://WWW.extempo.com/.

Finin, T., Fritzson, R. (1994). KQML as an Agent Communication Language. The Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), ACM Press.
This work is related to the DARPA knowledge Sharing Effort.
http://www.cs.umbc.edu/kqml/papers/kqml-acl-html/root2.html.

FIPA (1999). Foundation for Intelligent Physical Agents. http://www.fipa.org.

Firefly, (1999). http://www.firefly.com/.

Foner, L. (1993). What's an Agent, Anyway? A Sociological Case Study.
http://foner.www.media.mit.edu/people/foner/Julia/Julia.html.

Franklin, S. and Graessner, A. (1996). Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents. In Proceedings of the Third International Workshop of Agent Theories, Architectures and Languages. Springer-Verlag.
http://www.msci.memphis.edu/~franklin/AgentProg.html.

Gardner, E. (1996). Behind the Scenes. Standards Hold Key to Unleashing Agents. Internet.com. 1996-04-29.
http://www.internetworld.com/print/1996/04/29/news/unleashing.html.

Genesereth, M. R. and Ketchpel, S. P. (1994:a). Software agents. Communications of the ACM, 37(7):48 – 53.

Genesereth, M. R. and Ketchpel, S. P. (1994:b). Software agents. http://logic.stanford.edu/papers/agents.ps.

Genesereth, M., R. (1)(No date).
http://Logic.Stanford.EDU/kif/

General Magic (1998:a). Agent Technology.
http://www.generalmagic.com/technology/mobile_agent.html.

General Magic (1998:b). Mobile Agents White Paper.
http://www.generalmagic.com/technology/techwhitepaper.html.

Glushko, R., Tenebaum, J., and Meltzer, B. (1999). An XML Framework for Agent-based E-commerce. Communications of the ACM, March 1999/Vol. 42, No. 3.

Goerlich, R. C. (1997). http://philly.cyberloft.com/bgoerlic/eliza.htm.

Grand, S. (1997). Three observations that Changed my Life. IEEE Expert, November/December 1997.

Gruber, T. (1999). What is an Ontology? http://www-ksl.stanford.edu/kst/what-is-an-ontology.html.

Guttman, R., Moux, A. and Maes, P. (1998). Agent-mediated Electronic Commerce: A Survey. Software Agents Group, MIT Media Lab. http://ecommerce.media.mit.edu/papers/ker98.pdf.

Gustavsson, R. (1999). Agents with Power. Communications of the ACM, March 1999, Vol. 43, No. 3.

Haptek (1999). http://WWW.haptek.com/.

Hayes-Roth, B., Johnson, V., Gent, R. van, Wescourt, K. (1999). Staffing the Web with Interactive Characters. Communications of the ACM, March 1999, Vol. 43, No. 3.

Hultmark, C. (1998). Elektronisk handel och avtalsrätt. Norsteds Juridik AB. ISBN 91-39-20105-8 [In Swedish].

IBM (1998). IBM Aglet Software Development Kit home page. http://www.trl.ibm.co.jp/aglets.

Jackson Higgins, K. (No date). Your Agent is Calling. http://www.firstfloor.com/editorial/622close.html.

jango.excite (1999). http://jango.excite.com.

JATLite (1999). Java platform with support for KQML messaging. http://java.stanford.edu/java_agent/html/.

Josef Stefan Institute (1999). Eliza. http://www-ai.ijs.si/eliza/eliza.html.

Jörgensen, N. and Svensson, J. (1986). Nusvensk grammatik. Liber: Malmö Sweden. ISBN 91-38-61700-5 [In Swedish].

Kaiserslautern (1997). University of Kaiserslautern. http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/.

Keahey, K. (1998). A Brief Tutorial on CORBA. The Object Management Group. http://www.cs.indiana.edu/hyplan/kksiazek/tuto.html.

Knowledge Sharing Effort (1994). http://www.ksl.Stanford.EDU/ knowledge-sharing/papers/kse-overview.html.

Krogh, C. (1996). With a license to KILL –9. Slideshow on agent technology. http://www.informatics.sintef.no/~chk/krogh/ papers/dnd96a/tsld016.htm.

Lange, D. and Oshima, M. (1998). Programming and Deploying Java™ Mobile Agents with Aglets™. Computer & Engineering Publishing Group. ISBN: 0-201-32582-9.

Lange, D. and Oshima, M. (1999). Dispatch your agents; shut off your machine. Communications of the ACM, March 1999/Vol. 42, No. 3.

Lotus (1999). http://www.lotus.com.

MIT (1999). MIT Media Lab home-page. http://lcs.www.media.mit.edu/projects/amalthaea/.

Maes, P. (1994). Agents that reduce work and information overload. Communications of the ACM, 37(7):31 – 40.

Maes, P. (1995:a). Intelligent Software. Scientific American, Vol. 273, No. 3, pp. 84 – 86, Scientific American, Inc. http://pattie.www.media.mit.edu/people/pattie/ SciAm-95.html.

Maes, P. (1995:b). Artificial Life meets Entertainment: Lifelike Autonomous Agents. Special Issue on New Horizons of Commercial and Industrial AI, Vol. 38, No. 11, pp. 108 – 114, Communications of the ACM, ACM Press, November 1995. http://pattie.www.media.mit.edu/people/pattie/ CACM-95/alife-cacm95.html.

Maes, P., Guttman R., Moukas, A. (1999). Agents that buy and sell. Communications of the ACM, March 1999/Vol. 42, No 3.

Market Maker, (1999). http://maker.media.mit.edu/.

McCarthy, J. (1979). Ascribing mental qualities to machines. Technical report, Stanford University AI Lab., Stanford, CA 94305. http://www-formal.stanford.edu/jmc/ascribing.html.

Mchacko (1999). The Telescript Language Reference Version 1.0. http://science.gmu.edu/~mchacko/Telescript/docs/telescript.html, http://science.gmu.edu/~mchacko/Telescript/docs/TSLangRef_Preface_2.html.

Microsoft Cooperation (1999). http://www.microsoft.com/com/dcom.asp

MIT Media Lab, Agents, (1999). http://agents.www.media.mit.edu/groups/agents/.

Mitsubishi (1999). Mitsubishi Electric Information Technology Center America (ITA).http://www.meitca.com/HSL/Projects/Concordia.

Merriam-Webster (1999). Encyclopedia Britannica OnLine. Merriam-Webster's Collegiate Dictionary. http://members.eb.com.

Moore, D. (1998). Software Agents: Creating a Structure for Cyberspace. Web-site of Dana Moore. http://home.att.net/~dana.moore/agent_2.htm.

Moore, D., Paciorek, N., Wong, D. (1999). Java-based Mobile Agents. Communications of the ACM, March 1999, Volume 42, no. 3.

Moukas A. (1996), Amalthaea: Information Discovery and Filtering using a Multiagent Evolving Ecosystem, Proceedings of the Conference on Practical Application of Intelligent Agents & Multi-Agent Technology, London, 1996. http://lcs.www.media.mit.edu/projects/amalthaea/.

Mozart (1999). The Mozart Programming System. Homepage. Pages on Mozart: http://www.mozart-oz.org/, http://grizzly.ps.uni-sb.de/.

Mozart (1999:a). The Mozart Programming System. Tutorial of Oz: Introduction. http://www.mozart-oz.org/documentation/tutorial/node1.html#label1.

Mozart (1999:b). Site of Mozart-org. http://www.mozart-oz.org/download/.

My Yahoo (1999). http://my.yahoo.com.

ObjectSpace Voyager (1999). http://www.objectspace.com/products/voyager/index.html.

OMG (1998). CORBA Overview. http://www.infosys.tuwien.ac.at/Research/Corba/OMG/arch2.htm#446864.

OMG (1999): http://www.omg.com/.

OnSale (1999). http://www.onsale.com/.

Passagen, (1999). Web-site. http://www.passagen.se/.

Personalogic (1999). Web-site. http://www.personalogic.com/.

Petrie, C., J, (1996). Agent-Based Engineering, the Web, and Intelligence. IEEE Expert December 1996. http://cdr.stanford.edu/NextLink/Expert.html.

PointCast (1999). http://www.pointcast.com.

Rubin, A. D. and Geer, Jr., D. E. (1998). Mobile Code Security. IEEE Internet Computing, vol. 2 number 6 dec. 1998. Issue on Security and Mobile Code. IEEE Computer society Publications Office, Los Alamitos, CA, USA. Available on line for members of the Computer Society at http://computer.org/internet.

Shoham, Y. (1993). Agent-oriented programming. Artificial Intelligence, 60(1):51 – 92.

SICS (1999). Swedish Institute of Computer Science. Intelligent Systems Laboratory. http://www.sics.se/ps/.

SITI (1999). Tahuti-project. http://www.sisu.se/~alexandr/Tahuti/index.html

Sun Corporation (1999). http://java.sun.com/products/jdk/rmi/index.html.

Sycara, K. P. (1998). The Many faces of Agents. In AI Magazine – An official publication of the American Association of Artificial Intelligence Volume 19 No 2, summer 1998, 11–12. ISSN 0738-4602. http://www.aaai.org

Tacoma (1999). The TACOMA project (Tromsø And COrnell Moving Agents). http://www.tacoma.cs.uit.no/.

Tanenbaum, A. (1992). Modern Operating Systems, Prentice-Hall, New Jersey. ISBN 0-13-595752-4.

Tham, C., Friedman, B. (1996). Common Agent Platform Architecture. The Agent Society. http://www.agent.org/pub/satp/papers/satp.html.

Tete-à-Tete, (1999). http://ecommerce.media.mit.edu/tete-a-tete/.

UMBC (1993). UMBC KQML Web, Computer Science & Electrical Engineering, University of Maryland Baltimore County, Baltimore Maryland 21250 USA. http://www.cs.umbc.edu/kqml/kqmlspec/spec.html.

UMBC (1999:b) Agent Web, Knowledge Sharing Effort, http://WWW.cs.umbc.edu/agents/kse.shtml.

Virtual Personalities (1999). http://www.vperson.com/.

Verity (1999). http://www.verity.com.

W3C (1996). Mobile Code Systems. http://www.w3.org/MobileCode/

W3C (1999). Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation 22 February 1999. http://www.w3.org/TR/REC-rdf-syntax/.

Waldo, J., Wyant, G., Wollrath, A., Kendall, S. (1994). A Note on Distributed Computing. Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 USA. http://www.sunlabs.com/techrep/1994/abstract-29.html.

Waltzman, R. (1998). Industrial Applications of Artificial Intelligence. Lecture notes.

Weizenbaum, (1966). Eliza – A Computer Program for the study of natural communication between man and machine. Communications of the ACM, Vol. 9, No. 1, January 1966.

White, J. (1996). Prospectus for an Open Simple Agent Transfer Protocol. General Magic Inc. http://www.agent.org/pub/satp/prospectus3.html.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent Agents: Theory and Practice. In Knowledge Engineering Review Volume 10 No 2, June 1995, 115-152. © Cambridge University Press, 1995. http://gryphon.elec.qmw.ac.uk/dai/pubs/KER95/index.html.

Ygge, F. (1998). Market-Oriented Programming and its Application to Power Load Management. Ph.D. thesis. http://www.enersearch.se/~ygge/Work/Pubs/thesis/thesis.html.

11 Glossary

This glossary explains some terms in the paper. It is based on a glossary found at Electronic Commerce (1997).

AI
Artificial Intelligence.

Animism
From German Animismus, from Latin anima (= soul)

1. a doctrine that the vital principle of organic development is immaterial spirit
2. attribution of conscious life to objects in and phenomena of nature or to inanimate objects
3. belief in the existence of spirits separable from bodies

[Merriam-Webster, 1999]
Common in primitive religions.

API
Application Programming Interface.

ARPA
See DARPA.

CAP
Common Agent Platform. The proposed implementation of SATP*.

CPU
Central Processing Unit. The unit, most often an integrated circuit, that does the actual work in a computer.

Corba
Common Object Request Broker Architecture. The core of the Object Management Architecture defined by the Object Management Group, OMG*.

DAI
Distributed artificial intelligence.

DARPA
Defense Advanced Research Projects Agency. The central US Department of Defense research organization. Sometimes in it's history called ARPA. Responsible for breaking new ground in many areas of computer science. DARPA created ARPA-net, a computer network system later to be known as the Internet. http://www.darpa.mil/.

DCOM
A platform for distributed computing from Microsoft.

Demon of Laplace
Pierre-Simon Laplace (1749–1827), French mathematician, astronomer, and physicist. Laplace argued that a demon with complete knowledge of the positions and velocities of all the particles in the universe and infinite calculating capabilities would be able to describe the complete state of the universe at any point in time, past or coming.

EC
Electronic Commerce*.

EDI
Electronic Data Interchange*.

Electronic Commerce
Can be simply described as doing business electronically. More precisely it is conducting the exchange of information using a combination of structured messages (EDI), unstructured messages (Email), data, databases and database access across the entire range of networking technologies. The sharing of information with business partners leads to cost savings, increased competitiveness, improved customer relations and greater efficiency through the redesign of traditional processes.

Electronic Data Interchange (EDI)
Electronic data interchange (EDI) is the exchange of documents in a structured form between computers via telephone lines. It is being increasingly used to great effect worldwide, most commonly, but not exclusively, for purchasing and distribution – orders, confirmations, shipping papers and invoices – but also for dentists payments and the distribution of exam results.

Electronic Directories
A directory is quite simply a database containing information held in a logical structure. Key details about the users of messaging systems are kept here, most importantly their address. X.500 Directories can be distributed, residing on several computer systems, perhaps spread over a number of sites. However the combined information can be accessed from a single point. This has numerous advantages for organizations not least the additional processing power. Furthermore, if one machine goes down, the directories continue to work.
The information structure behind an X.500 Directory is called the Directory Information Tree (DIT) where information is held in hierarchical form as required by the organization. It may start with countries at the top, followed by organizations, then departments and finally individuals.

Electronic purses
Using smart card technology an electronic purse is created with cash stored electronically on a microchip, creating a pre-payment card, which can then be used to buy a range of goods and services. This allows the safe transfer of value to another electronic purse. Trials are underway in many countries – in the UK, the Mondex trial started in 1995.

Email
Electronic Mail (Email) is the electronic exchange of unstructured information. Users can send, receive, forward and store text messages or chunks of data quickly and easily, regardless of time zone or geographic location. It is fast becoming a quick and efficient alternative to more traditional forms of communications such as memos, facsimiles, telephone, voice mall, the postal system and even face to face meetings.

Extranet
An Internet-based virtual network joining the Intranets of different enterprises together, making a collaborative inter-enterprise electronic community.

FEDI
Financial Electronic Data Interchange (FEDI) involves the computer to computer transmission of both payment instructions and remittance details using international message standards. An example would be trade payments – e.g. a retailer sending a payment to a supplier in payment of multiple invoices.

Gestalt
A structure, configuration, or pattern of physical, biological, or psychological phenomena so integrated as to constitute a functional unit with properties not derivable by summation of its parts. [Merriam-Webster 1999]

Hybrid EDI
Introduced by service providers to accommodate situations in which only one trading partner is capable of using EDI, while the other continues to trade using traditional methods involving paper or fax. An example would be a trading partner sending an electronic purchase order that is then faxed by a service provider to the recipient.

IDL files
Interface Definition Language. ISO standard no. 14750. Commonly referring to the OMG IDL.
Descriptive title: Information technology—Open Distributed Processing—Interface Definition Language
Source: Richard Mark Soley <soley@omg.org>

IIOP (Internet Inter-ORB Protocol)
A transport protocol which is part of the Object Management Group's Common Object Request Broker Architecture (CORBA) for distributed computing.

Internet
A global network of networks, it provides connections for sending electronic mail messages, transferring files, linking to other computers and accessing information available in a variety of different forms, such as bulletin boards for people with common interests or electronic product catalogues.

Internet Cash
Purchased from an issuer (bank or credit institution) and then exchanged freely over the Internet. It is aimed at low value payments, both cross border and domestic. Internet cash will be bought in local currency, with the buyer then sending the ecash to the seller in an Internet message.

Interpreter
An interpreted computer language like Oz, Lisp or Scheme, is not run directly "on the CPU", but is run by an interpreter or virtual machine (VM for short), a program, that is executed on the computer. Programs written in an interpreted language can be made platform independent if interpreters are installed on the machines on which the programs are to be run. The interpreters in their turn are for the most part platform dependent. In the Java language the source code is compiled to "bytecode" that is subsequently run on the Java interpreter, the Java Virtual machine.

Intranet
Narrowly, it is the use of low-cost Internet technologies to create internal information networks. More broadly, it spans an organization's entire information network, including the use of Internet technologies as well as PC-to-host connectivity, mobile communications, client/server networks and integration of data warehouses, for example.

IPR
Intellectual Property Rights.

Kernel
The central part of the operating system.

KIF
Knowledge Interchange Format. The knowledge representation format used in the inter-agent communication language of ARPA's Knowledge Sharing Effort. Based on first order predicate logic.

Knowledge Sharing Effort
An ARPA project aiming at reusable, component-based, distributed knowledge systems.

KQML
The “outer” language of ARPA’s KSE. A language and a set of protocols that support computer programs in identifying, connecting with and exchanging information with other programs.

KSE
See Knowledge Sharing Effort.

MAS
Multi agent system*.

Micropayments
A system for paying small amounts of money directly from your own computer, or via a (mobile) software agent.

MIME
Multipurpose Internet Mail Extensions. An e-mail format capable of handling non-English characters, pictures and video.

MUD
Multi User Domain (Multi User Dungeon). A normally text-based web-interfaced virtual reality. Users can log in, chat with other users currently on the domain and move between different localities, all by text IOU.

Multi Agent System
Agents cooperate in a system in order to solve a common problem or to serve their own best interests.

Multimedia
Multimedia represents the merging of the computer, communications and broadcasting industries. By combining a variety of information sources, such as voice, graphics, animation, images, audio and video in an exciting and highly dynamic medium, multimedia systems will revolutionize the way we work, learn and play.

ORB
Object Request Broker.

OO
Object Oriented.

Object Request Broker
ORB in short. Communication infrastructure between clients issuing a request and servers/objects responding to these requests. Most often referring to CORBA-compliant ORBs.

OMG

Object Management Group. A non-profit corporation, the Object Management Group (OMG) was founded in 1989 by eight companies: 3Com Corporation, American Airlines, Canon, Inc., Data General, Hewlett-Packard, Philips Telecommunications N.V., Sun Microsystems and Unisys Corporation. Through the OMG's commitment to developing technically excellent, commercially viable and vendor independent specifications for the software industry, the consortium now includes over 800 members. OMG is the source of the Corba* specification. http://www.omg.org/.

Online Catalogue

An online catalogue is basically a web site that allows products to be viewed and ordered online. To help with shopping online, the customer needs an electronic trolley to wheel round the catalogue and load up with shopping.

PDF

Portable Document Format. PDF is a universal electronic file format. The PDF format is modeled after the PostScript language and is device- and resolution-independent. Documents in the PDF format can be viewed, navigated, and printed from any computer regardless of the fonts or software programs used to create the original. PDF files can be delivered as many times as needed over any electronic medium – the Web, CD-ROM, network servers, on-line services, email, and more. http://web7.whs.osd.mil/corres/viewing/pdfdef.htm.

PDI

Personal digital assistant. Normally an agent appearing at your workstation environment.

Purchasing card

Aimed at the business market, the purchasing card allows company staff to deal directly with suppliers and reduce costs by cutting out paper, e.g. the need for a purchase order. Orders can be placed over the phone and the company receives management information detailing spending by employee, supplier etc.

RDF

Resource Description Framework. A meta-data system. http://www.w3.org/TR/REC-rdf-syntax/.

Remote Method Invocation

Cf. RMI.

RMI

Remote Method Invocation in Java. An Object Oriented version of RPC. Member functions and procedures of objects are called methods in the OO* world. http://java.sun.com/products/jdk/rmi/index.html.

$\mathbf{R}^n$

Mathematics. **R** are the real numbers. $\mathbf{R}^2$ are the real numbers in two dimensions, e.g. the points in a plane. $\mathbf{R}^3$ are the real numbers in three dimensions, e.g. the points in three-dimensional space. $\mathbf{R}^n$ are the real numbers in n dimensions, where n stands for any integer bigger than zero.

RPC

Remote Procedure Call. A design solution that makes inter-process/inter-machine communication transparent to the programmer. A call to a procedure on another machine looks like an ordinary procedure call. See [Tanenbaum 1992].

SATP

Simple Agent Transfer Protocol. A “http” for agents, proposed to W3C in 1996 by James White and others. To be implemented by CAP, the Common Agent Platform.

SICS

Swedish Institute of Computer Science.

Smart cards

A plastic card incorporating an embedded microchip - a tiny computer. In extensive use in France and Germany.

Tacoma

The TACOMA project (Tromsø And COrnell Moving Agents). http://www.tacoma.cs.uit.no/.

Tcl

Tcl (pronounced Tickle) is a general purpose programming language originally intended to be embedded in other applications as a configuration and extension language. Also used in mobile agent systems Agent Tcl and Tacoma. The success of one its most important embeddings, the Tk toolkit for the X Windows System, has resulted in Tcl and Tk together being most heavily used for building graphical user interfaces (GUIs). It is also heavily used as a scripting language like Awk, Perl or Rexx, and (as Expect) is used to script interactive applications (e.g., to automate telnet logins to various information vendors). http://www.lib.uchicago.edu/keith/tcl-course/tcl-course.html.

Theodicy
Modification of French théodicée, from Latin theo (= god) and Greek dike (= judgment). Defense of God's goodness and omnipotence in view of the existence of evil.

Virtual Machine
Same as interpreter*.

VM
Virtual Machine*.

W3C
World Wide Web Consortium. The W3C is a world wide industrial consortium. It was founded in 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C is jointly hosted by the Massachusetts Institute of Technology Laboratory for Computer Science [MIT/LCS] in the United States; the Institut National de Recherche en Informatique et en Automatique [INRIA] in Europe; and the Keio University Shonan Fujisawa Campus in Japan. http://www.w3.org/.

Watermarking
Cryptological techniques for placing meta-information in a text- or image document. This information can contain data on IPR issues and also used for making the document useless of these IPR rights are violated.

World Wide Web (WWW)
Currently the fastest-growing aspect of the Internet, it allows information to be accessed by subject matter regardless of its location – a real advantage in a network as vast and complex as the Internet. Users move automatically from one database (or site) of interest to another using “hyperlinks”. Increasing levels of complexity enable interactive, multimedia facilities to be developed.

X.400
A set of internationally-agreed recommendations describing a standard approach to building messaging Systems which can be used to carry Email, EDI, fax and a range of other data. X.400 boasts a number of key features not available on Internet mail systems, including notifications confirming the delivery or non-delivery of messages and security.

X.500

Comparable to an electronic yellow pages where a wide variety of data can be stored – names, Email and postal addresses, phone numbers, even photos and video clips. Information on machines and the routing of Email can also be stored here and used by a message handling system. X.500 is also a series of internationally agreed standards detailing how to build such a directory. It is most frequently used as a corporate address book, but has the potential to become a global source of information.

XML

Extensible Markup Language XML is a grammar for creating mark-up languages, so called XML applications. One such XML application is RDF, Resource Description Framework. XML is supported by W3C. http://www.w3.org/XML/.