ESPRIT Project No. 25 338

Work packages D and F

Deliverables DD1 and DF1

Survey of Autonomous Agents and Service Interaction

ID:	DD1, DF1	Date:	23.10.97
Author(s):	Nick Taylor, Steve Battle	Status:	Draft
Reviewer(s):		Distribution:	

Change History

Document Code	Change Description	Author	Date
DD1, DF1	First version of document. No changes.	Nick Taylor Steve Battle	1.12.97

1 Introduction

In recent years, the term "agent" has been used more and more to describe a multitude of computing related activites up to the point now where it has probably earned a position in the buzzword "hall of fame". Researchers, commentators and vendors are to some extent, all to blame for the current confusion surrounding the role of agents in emerging architectures and applications. One of the aims of this document is to provide a definition of an agent that is non-contentious and borrows heavily from the human notion of an agent, and outline how this notion of agency is currently being applied. In addition, we will also survey current scripting technologies as a means of defining the goals and behaviour of agents.

It should be noted that this survey will focus on the area of mobile agents and not intelligent agents. FollowMe is concerned with issues of mobility and therefore a survey of intelligent agents is beyond the scope of this document

This document will combine the surveys of workpackages D (Autonomous Agents) and F (Service Interaction) into the same document since they are connected, however, the Autonomous Agents survey will concentrate on mobility whereas the Service Interaction survey will look at programmable interfaces to services, and associated issues.

2 Autonomous Agents

2.1 Software Agents

We are all familiar with the human notion of an agent, some of us come into contact with them every day. Consider an estate agent; they act on behalf of other parties in order to facilitate the buying or selling of a house. The key here is "act on behalf" which implies an act of delegation from the client to the estate agent. The purchase client delegates the activity of finding suitable properties to the agent who then selects suitable candidates from an available pool, according to the criteria specified by the buyer, and then presents these to the client. In the case of a selling client, the agent takes the details of the property and makes them available to purchasing clients. Sometimes the agent will charge a fee for doing something on behalf of a client; in the estate agent case a selling client is usually charged by the estate agent in the form of a percentage of the selling price, however, a buying client is not normally charged.

The computational agent is very similar to the human notion of an agent in the sense that an entity delegates an activity to the agent which is then expected to carry out that activity. It may also be the case that the agent charges a fee of some description but this is not always the case. A software agent, much like it's human equivalent, has to prove it's competence in being able to represent a client's interests and thereby gain the user's trust; two issues discussed by Patti Maes in [1].

2.2 Definition of an Agent

For the purpose of this survey, we will adopt a "weak" or uncontentious definition of a computational or software agent as described in [2] and suggests that an agent is:

· ***autonomous***
An agent has it's own goals and mechanisms to achieve those goals.

· ***social***
An agent can interact with other agents and humans through well defined interfaces.

· ***reactive***
An agent is able to percieve it's environment and respond to changes in it.

· ***proactive***
An agent is able to not only react to it's environment but also to demonstrate goal directed behaviour and can take the initiative where possible to achieve it's goals.

As was mentioned earlier, these four abilities define a weak notion of an agent or the basic features most definitions include. There are other ideas an agent can embody that are more contentious such as veracity (the assumption that an agent will not knowingly give false information), benevolence (the assumption that agents do not have conflicting goals and will always try to do what is asked of it) and rationality (the assumption that an agent will act in order to achieve it's goals). For a more detailed discussion on the "strong" definitions of agents, see [3].

2.3 Agents and Mobility

Agents which are capable of travelling from machine to machine via network are referred to as Mobile Agents. Mobile agents embody the four ideas mentioned earlier and add the ability to carry their state and code with them when they move. This leads to a new model for distributed computing where the clients, which are typically static in peer-to-peer and client-server models, have the ability to move between “places” which are abstractions of the physical machines in the network. This allows an agent to migrate to the service (or agent) they are interested in interacting with, and so reduce network traffic during the interaction with the agent needing only to report salient events (if indeed there are any) back over the network to an interested party.

It has been suggested that mobile agents are a technology waiting for an application which was probably true when mobile agents were first conceived. It has also been suggested that there are no applications which agents implement that can't be implemented using other technologies which is also true. Danny Lange (former Aglet developer, now with General Magic), when asked if there any problems that agents and agents alone were capable of solving replied "Are there any problems that object and objects alone are can solve? No...most people prefer to use objects because they provide better abstraction..." [4]. This idea is gaining a lot of momentum and may quite possibly be the next evolution in system conception and design. We are now seeing applications being proposed and prototyped which use agent and mobile agent technology to deliver real benefit to the user through a more intuitive agent based interpretation of the problem domain.

2.4 Mobile Agent Systems

An agent system in this context is a development environment which allows the construction of applications based upon agents. The first commercially available agent system was Telescript from General Magic [5]. Telescript allowed mobile agents to be created and deployed in abstractions of the physical host known as “places”. The agents could interact with each other and utilise the built in security features for authentication. Although systems using Telescript were deployed, the system never really caught on mainly due to the proprietary language agents were written in and the cost. Then along came Java which, due to the combination of an easy to use language, virtual machine and byte code definition, fueled a number of agent system development initiatives and added great value through the freely available Java Deveopment Kit (JDK) and the numerous platforms supported. One of the first Java based agent systems to emerge was MOLE [6] from the University of Stuttgart; it provides basic facilities such as the concepts of locations, agents, migration of agents between locations, remote execution of methods and messaging.

Many other agent systems are available presently, and more seem to appearing on an almost daily basis. Some of the more well known systems are Aglets from IBM [7], Voyager from ObjectSpace [8], Concordia from Mitubushi [9] and Odyssey from General Magic [10]. These systems all support the notion of agents to some degree and some systems provide services such as security and persistence which agents may use, however, as with many new technologies, the “getting your hands” on the technology preceeds any prescribed methods of employing it. Since the systems mentioned earlier are based on Java and therefore support the implementation phase of object oriented methodologies, current methods are aimed at static objects and include no tools to model objects which may migrate and the reasons why they may migrate. These deficiancies will need to be rectified before any potential of mobile agents will be recognised.

Efforts are under way to standardise the interfaces and components of agent systems with the OMG issuing request for proposals for a Mobile Agent Facility (MAF) [11]. So far, one response from a group of interested parties including Crystaliz, General Magic, GMD FOKUS and IBM has been received however it is still in a draft form. Although this response is strong in the vocabulary and abstractions it specifies (hardware and software) it is not clear how issues such as diversity in agent facilities and agent migration will be resolved. It is not mature enough to use as a target specification for FollowMe autonomous agents.

2.5 Applications of Mobile Agents

There is a great deal of research being conducted into the applications of mobile agents (see [12] for more detailed information) and below are some of the areas of interest.

2.5.1 Information Retreival

According to Maes [1], the "information overload" situation is with us; there is just so much unstructured information available that we are in danger of being swamped. Maes has suggested that agents could be one way one way of avoiding this crisis. Offline retreival is a promising application area that allows a user to delegate the search for information to a software agent which is responsible for filtering out inappropriate information while being proactive by discovering new information sources. More sophisticated systems would allow the search to continue while the user is not contected to the network or present at a machine. A typical approach would be to send an agent off into the network with a task of gathering information on a topic, disconnecting and waiting for the agent to complete the activity. This approach can allow a user to be more productive while the search is in progress and also money through not having to be connected for the duration of the search.

2.5.2 Monitoring Remote Resources

Managment of remote resources is currently under investigation by a number of hardware centric companies such as Hewlett Packard. They are interested in sending agents to remote devices in order to to monitor status of the device and signal failures to the administrator.

2.5.3 Market places and Auctions

Market places are meeting places for buying and selling agents [13]. Buying agents are interested in purchasing something on behalf of their owner and selling agents are interested selling something belonging to their owner. The market place facilitates the interaction between agents allowing buying agents announce their wish to purchase something, new selling agents to be introduced to existing buyers and the subsequent negotiation between buying and selling agents in order to strike a deal. The market place defines the language protocol that is used and so long as the agent knows the language, they can participate in conversations. Auctions are different to Markets in that they use formal bidding protocols for price negotiation which all participating agents must be aware of, such as Dutch and Vickrey auction protocols. There are many prototype auction houses currently be researched, an example is [14] which is a Java based framework investigating the fish market bidding protocol.

2.6 The "Personal Assistant" Metaphor

The "personal assistant" (PA) metaphor has arisen from a need to help computer users deal with the increasing complexity of the environments and systems they interact with. They are essentially interfaces which are personalised for a particular user, interact collaboratively with their user and learn or gain competance from their user by watching (or being taught) how the user accomplishes a task. They may also be proactive and can offer (or be told) to take responsibility for certain tasks which may be bringing to the user's attention an email message that should be read immediately or filtering out news messages which will not be of interest to the user.

Much research work has been done in the area interface agents and personal assistants ranging from the basic properties they may possess to virtual reality based visions [15]. PAs may be modelled as agents that present a familiar and personalised interface to their user and assist the user by performing apprpriate tasks so freeing the up the user's time. Much of the thrust of the research into PAs has been into intelligent assistants; these learn a user’s habits by watching what they do at their machines through a variety of techniques including user programming, knowledge engineering or machine learning techniques to assist the user in dealing with the various applications they interact with. There are many examples of agents that learn their user's preferences such as the elecronic mail agent described in [16] to the Microsoft Agent [17] which is essentially framework within which a programmer can create animated characters that react to a user's input.

Having said that much work had be done on interface agents and personal assistants, not much work has been done in the area of network based PAs which is the area that FollowMe will focus on. Within FollowMe, a PA may have to interact with the user via a number of devices so that the user can initiate tasks and receive the results of those tasks in a number of different environments. It must be highly available to the task agents it has dispatched so it can continue it's coordination however the user may well be disconnected from it for long periods of time. For example, a user travelling in their car could dial up their PA via a hands-free cellphone and have a personalised selection of news articles read out. This could be viewed as an extension of the Personal Assistant metaphor; instead of the PA residing close to the user at all times, the PA has the ability to migrate into the network when the user is disconnected where it can remain available to the tasks it has launched and be contactable by agents for mediation.

2.7 Agent Languages

The subject of `agents' has probably generated more special purpose languages than any other area of computer science. These range from languages with a *strong* commitment to agent intelligence, particularly with respect to anthropomorphic concepts of belief, desire and intention, to so-called *weaker* languages with a greater emphasis on the software engineering. As the aim of the followMe project is not to develop *intelligent* agents, but *mobile* agents, this document will only include discussion of languages which are weak with respect to concepts of agency. For an in depth review of strong agent languages the reader is referred to [2]. While this may appear to be a great loss, the shift to more object oriented languages means that the real functionality of the agent is captured in the components which make it up, rather than in the trivialities of the syntactic sugar used to glue them together.

The role of the agent description language is to tie together the various components that make up an agent, and to shield the user from details of the underlying mechanisms. Agent languages are scripting languages typically with features (or components) supporting mobility and communication. In the context of *mobile* agents, the agent designer should not be concerned with the suspension and resumption of threads as an agent is moved from one location to another. While the designer requires the power of a real programming language, we would also want to offer a flexible prototyping system that many compiled languages fail to deliver. This is the realm of so-called scripting languages. Bear in mind that the greatest benefit of scripting within followMe lies not in its functionality but in its *extensibility*; the ability to add new functionality, new agents, without having to recompile.

2.7.1 Scripting Languages

Scripting languages can be distinguished from conventional programming languages along a number of different axes; interpretetation versus compilation, strong versus weak typing, low-level versus high-level primitives, static versus dynamic allocation, and their limited support for object-orientation and threading.

Scripting languages don't require a separate compilation phase before execution. The aim is to reduce the time spent going round the traditional development cycle (edit-compile-execute), the script writer need only edit and execute. Although this means scripts are less efficient than their compiled cousins, this is not a problem as they shouldn't be used for time consuming, computer intensive tasks.

Conventional programming languages are increasingly strongly typed, ensuring that data isn't lost in accidental type conversions. This places the burden on the programmer to explicitly indicate where valid type conversions can occur. By comparison, scripting languages are generally weakly typed, trading rigour for ease of use. These factors mean that the development time of a scripted application is shorter than with traditional approaches.

With the increasing use of inter- and intra-net based systems, scripting languages have become increasingly important as a means to connecting together different system components. Whereas programming languages like Java are designed to build systems from the ground up, scripting languages generally assume the existence of the necessary high-level objects. The script provides a quick and dirty way of connecting these objects together, acting as a kind of "component glue". For example, the Tool Command Language (TCL) is used to arrange the layout of GUI components within a window, and Unix shell scripts assemble a number of different processes into a pipeline. Scripting languages and programming languages are complementary in this respect.

With compiled languages, it is efficient to use statically defined structures wherever possible. This applies to every feature of the language; including data declarations, function or method definitions, function or method declarations,

and class definitions. The variables in a scripting language are generally highly dynamic structures. For example, whereas Java provides both fixed and variable length strings, only dynamic strings are found in JavaScript, simplifying the choice of primitive data types. In a loosely object-oriented scripting language such as JavaScript even the classes themselves are created on the fly. Scripting languages also typically have good support for garbage collection, saving the user from the headaches of discarding unused data and objects. This feature is by no means unique to scripting as Java also provides good support for garbage collection.

Like most other computer systems, scripting languages are becoming increasingly object-oriented. As noted above, JavaScript provides very basic support for the definition of classes and the creation of objects. The JavaScript language provides no built in class inheritance mechanism, although again the same effect can be explicitly created by the programmer in the dynamic construction of an object. This is a deliberate omission as inheritance can be seen as having as having a negative effect on object encapsulation; exposing the innards of a class to its subclasses. These dependencies ultimately reduce the potential for re-use of script fragments.

The complexity of the threading model for scripts is kept to a minimum. Whereas conventional programming languages like Java provide a sophisticated multi-threaded model, most scripting languages stick to a single thread. Because of the increasing use of scripts to define user-interfaces, as in TCL/TK and JavaScript, the only complication of this model is the ability of scripts to respond to user events. Though there are similarities between threads and event handlers, the latter should be short-lived so we generally don't need to worry about problems like synchronization and deadlock which plague concurrent systems.

2.7.2 Scripting Architectures

It is important to distinguish between a scripting language and its underlying scripting architecture. The scripting architecture is a foundation comprising the basic services available to the scripting language. For example, AppleScript is built on top of the Open Scripting Architecture which provides basic event based messaging, and an interface repository. The advantage of this division is that different vendors are free to introduce their own scripting languages providing they are compliant with this architecture.The spirit of interoperability is one of the main objectives of the Object Management Group (OMG), whose proposed CORBA component model RFP (orbos/97-06-12) class for an architecture that maps directly onto JavaBeans. The main points of interest are again event based messaging, with *introspection* as the means of determining an object’s interface.

2.7.3 Scripting the Desktop

Although simple scripting can be seen in the Job Control Languages of early mainframe systems, it is on the desktop that they have gained in popularity. The early Macintosh user interface sported a ground-breaking graphical user interface, yet for years it lacked any principled mapping onto a well-defined language. With the development of AppleScript, GUI operations could be recorded and played back to control interface operations. CORBA has made few inroads onto the desktop, which may be due in part to the fact that CORBA currently provides no scripting language support. The CORBA scripting language RFP (ORBOS RFP9) calls for a langauge that fits naturally into the proposed CORBA component model. It is likely that either Netscape’s JavaScript or Sun’s Jacl (The language formerly known as TCL), or both, will be conscripted into this role. With stiff competition from DCOM/ActiveX supported by scripting in the form of Visual Basic, Microsoft’s domination of the desktop will be hard to break. Visual Basic has eveolved from its humble beginnings as Microsoft Basic to a flexible and all-encompassing scripting language. As a consequence, the kind of people writing programs is no longer restricted to trained programmers, but now includes a new breed of end-user programmers. Scripting languages are ideally suited to the demands of these new users.

2.7.4 Scripting the Web

Web based distribution is currently based on the client-server model. The norm today is for the server to deliver content in the HyperText Markup Language (HTML), or it may provide services via the Common Gateway Interface (CGI). CGI back-ends are increasingly being written in the Perl scripting language. While CORBA is weak on scripting, it offers a thoroughly comprehensive framework for service interaction.

2.7.5 Scripting for agents

Agent based computing breaks out of the traditional client-server model, allowing peer-to-peer interaction - the object web. Mobile agents demand additional flexibility in allowing processes to jump from one machine to another. With this cross platform capability, the script interpreter must be available on every platform the agent may arrive at. This may be a native interpreter as in Netscape's current implementation of JavaScript, or the interpreter itself may be cross-platform in that it runs within the Java Virtual Machine (JVM), as with Sun's Jacl. In the case of mobile agents, it is important that the script interpreter provides support for the suspension and resumption of a script. The saved state, or *continuation*, must be serializable for transmission. Ideally, this would be transparent to the running script.

2.7.6 Comparisons

The table below evaluates the current state of play for a variety of scripting languages in terms of the environments they work in. The environments included are various desktops, web clients and servers, and agent based architectures. Both Tcl backed by Sun, and JavaScript backed by Netscape, are strong contenders in the CORBA scripting RFP.

Language	desktop friendly	client friendly	server friendly	agent friendly
Tcl	Y	Safe-Tcl		Agent-Tcl
AppleScript	Y			
Visual Basic	Y	VBScript		
Rexx	Y	NetRexx	Y	
Unix shell	Y			
Python	Y			
TeleScript				Y
Perl		Y	Y	
ACTOR				Y
JavaScript		Y	Y	?

The only existing scripting langauges with agent-friendly implementations are Tcl, TeleScript and ACTOR. Of these, TeleScript was the only commercially oriented product but has so far met with little success, other than contributing the concept of *places* to agent lore. ACTOR is quite an old language now and supports a style of massively concurrent computing which is inappropriate for the internet. Tcl is a very strong contender in the agent world, especially since its reincarnation as Jacl, the Java Command Language. Despite the non-existence of a specifically agent friendly version, JavaScript shouldn't be ruled out. In terms of user familiarity JavaScript wins hands down, and since its adoption by the ECMA (www.ecma.ch), JavaScript is now an open standard.

3 Service Interaction

The vision of agents roaming around the internet, utilising services on it's way has been proposed by many commentators. The services currently available on the internet unfortunately, are geared towards interaction with a human user and do not lend themselves to being used easily by a software entity such as an agent. To allow agents to interact with internet services, and ignoring securing issues for now, they must not only provide support for low-level interaction (at operation level) but also provide some high-level description of the service which must include how to use the service (it's behaviour) and a synopsis of what the service does. As we shall see, the low-level issues have been solved to some extent however the high-level issues have not and have been avoided due to the complexity.

3.1 Service Discovery

For agents to use a service, they must be able to discover it's existence. This implies that some kind of directory must be available to the agent which it can consult and discover services that may help satisfy the agent's goals. Trading as originally specified by the ODP-Trader standard (a good introduction is [18]) is a mechanism that allows service offers to be advertised, clients to browse the list of offers and select a suitable offer. See figure 1:

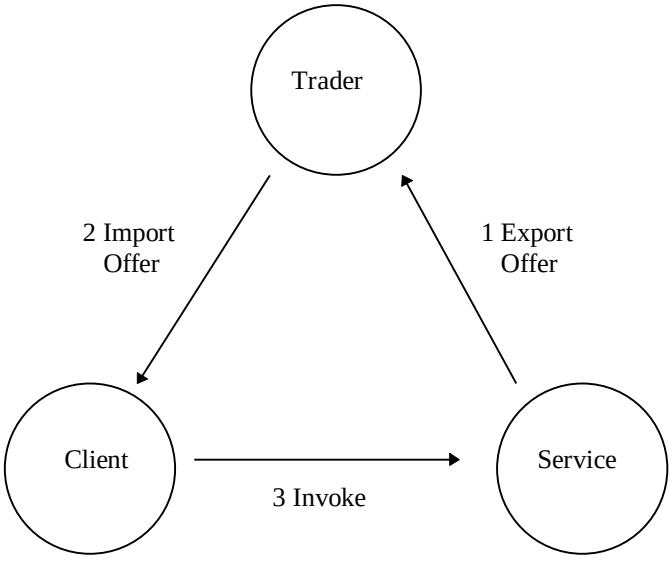

Figure 1

In figure 1, the service exports it's offer of service to the trader which stores the offer in it's offer database. The client, when it wishes to use a service, sends a request to the trader for a service of the desired type, and imports the offer. The client is then able to make invovations on the service. This allows clients to be decoupled from services, and allows new services to be dynamically added to a system.

Trading allows services to specify properties, for example, a persistency service may charge 5 ecu per kb of storage whereas another persistency service may charge 1 ecu per kb. Here, the price per kb is a property of the service. These properties allow clients to issue requests to a trader of the form “Find me a persistency service where price per kb < 3 ecu”, the trader discriminates on the basis of the price per kb property and returns a reference to latter service. The notion of properties is a powerful concept and may well be required in FollowMe.

The CORBA specification for Trading has been implemented by some vendors including Iona, and could be used as a basis for directory type services. Another alternative is the Java Naming and Directory service specification which offer a similar mode of operation as a trader and seamless integration with the Java environment.

3.2 Service Definition

In order for agents to be of use, they must be able to interact with services (for the purposes of this discussion, we will assume that an agent *is* capable of interacting with a service once the service is defined, no matter how). This means the agent must be able to find the service in order to do something on the user's behalf. The problem of how services present the operations they can perform has been solved by using declarative languages known as "interface definition languages" (IDL). These IDLs are precise specifications down to the number and types of parameter an operation takes and returns. Many example of IDLs exist such as Distributed Computing Environment (DCE) IDL, Microsoft IDL, Java Remote Method Invocation (RMI) and CORBA IDL.

The most widely accepted IDL is probably that of CORBA (Common Object Request Broker Architecture). CORBA is a standard [19] originating from the Object Management Group (OMG) that includes a language independant syntax for describing service operations. The OMG also specifies a number of language mappings (IDL to C++, IDL to Java for example) which allow legacy systems to be integrated into a CORBA based framework by using a technique known as encapsulation. This involves specifying an interface for the legacy in IDL, then using an appropriate mapping, implement the service by interfacing the legacy code with the code obtained from converting the IDL to the appropriate language.

The encapsulation approach could well be useful for the Service Interaction workpackage which is concerned with how services are specified. Legacy systems are a fact of life and many enterprises making the transition to an integrated IT strategy want to retain the investment in the legacy. Legacy code may well be part of a service to be provided in which case the CORBA approach could be the best way to specify services.

If a FollowMe system were to operate in a pure Java environment, Java RMI would be a viable option. Java RMI allows the calling of methods on remote objects and since it is part of the Java 1.1 API Specification, there is substantial support for it in the Java language. However, since JavaSoft has announced plans to implement the RMI protocol over the Internet Inter-orb Operability Protocol (IIOP), there is a good argument to allow both types of service definition in a FollowMe system although information regarding the service will have to be maintained that define the protocol to be used when communication with the service.

In general, IDLs are good at specifying the functionality of a service, however, their short-comings are that they do not describe how a service may be used eg operation1 must be invoked after operation2 and before operation3. This is a problem when services are discovered dynamically since the client has no way of knowing how the service may be utilised or even what the service does.

References

[1] Agents That Reduce Work and Information Overload, Maes P, Communications of the ACM, Vol 37, No 7, pp 31-40

[2] Software Agents, Wooldridge and Jennings, IEE Review, Jan 1996, pp 17-20

[3] Is it an agent, or just a Program?: A Taxonomy for Autonomous Agents, Franklin and Graesser, 3rd International Workshop on Agent Thoeries, Architectures and Languages, Springer-Verlag, 1997, pp 21-35

[4] http://computer.org/internet/online/v1n4/round.htm, IC Online Virtual Roundtable, The Future of Software Agents

[5] Mobile Agents, White, Software Agents, ed Bradshaw, pp 437-472, MIT Press, ISBN 0-262-52234-9

[6] MOLE, http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html

[7] Aglets Work Bench, http://www.trl.ibm.co.jp/aglets/

[8] Voyager, http://www.objectspace.com/

[9] Concordia, http://www.meitca.com/HSL/Projects/Concordia/Welcome.html

[10] Odyssey, http://www.genmagic.com/

[11] Mobile Agent Facility, http://www.genmagic.com/agents/MAF

[12] UMBC AgentWeb, http:/www.cs.umbc.edu/agents

[13] Kasbah: An Agent Marketplacefor Buying and Selling Goods, Chavez and Maes, Proceedings of the 1st International Conference on the Practical Application of Intelligent Agent and Multi-agent Technology, pp 75-90

[14] FM96.6: A Java-based Electronic Auction House, Rodriguez et al, Proceedings of the 2nd International Conference on the Practical Application of Intelligent Agent and Multi-agent Technology, pp 207-224

[15] The Virtual Tricorder, Wloka and Green, Brown University, Dept of Computer Science, 1995, CS-95-05

[16] Collaborative Interface Agents, Lashkari, Metral and Maes, Proceeding of AAAI 1994

[17] Microsoft Agent, http://www.microsoft.com/intdev/agent

[18] ODP-Trader, Bearman, International Conference on ODP, September 1993

[19] CORBA 2 Specification, Object Management Group, http://www.omg.org

Further reading on Scripting:

John K. Ousterhout, "Scripting: Higher Level Programming for the 21st Century", http://www.sunlabs.com/~ouster/scripting.html

John K. Ousterhout, "Scripts and Agents: The New Software High Ground", http://www.sunlabs.com/~ouster/agent.ps

John K. Ousterhout, "Why Threads are a Bad Idea (for most purposes)", http://www.sunlabs.com/~ouster/threads.ps

OMG, CORBA Scripting Language Request For Proposal, OMG Document: orbos/97-06-13, http://www.omg.org/library/schedule/ORBOS_RFP9.htm