An Overview of Agent Technology

"Perhaps there is something more to an agent than its capacity for beliefs and desires, but whatever that thing is, it admits no unified account within cognitive science."
(Shardlow, 1990)

Introduction

Since the late 1980s, researchers have shown an intense increase in interest regarding the use of software agents, in order to provide a powerful new approach for exploiting distributed systems (eg. Shoham, 1993; Genesereth and Ketchpel, 1994), although the seeds for such ideas were sown as early as the 1950s (McCarthy, 1959). Agents are commonly conceptualised as kinds of self-contained, concurrently executing software processes that encapsulate some state and are able to communicate with each other by exchanging messages. However, it should be noted that many different projects use the term "agents" for somewhat different entities, ranging from simple shell programs to systems incorporating real-time planning and other sophisticated capabilities.

The following senarios exemplify some typical uses of agents: such examples have motivated the use and development of software agents, even if current technology does not yet enable the provision of all such facilities to an appropriate standard.
A user wants to locate a published article, but only knows the title and author. The personal agent (PA) is directed to find the article. The PA knows how to find some papers on its local site, but when unsuccessful, it knows other useful ftp sites and also the addresses of specialist mediator agents. By contacting other agents, which in turn can contact still more agents, the PA will hopefully find an ftp site with a copy of the requested paper. It then arranges a transfer and output of the document, without direct personal supervision. Other routine processes can be automated in a similar way.
Upon logging in to their computer, a user is presented with a list of email messages, sorted into order of importance by their PA, and also a list of news articles. The PA highlights one particular article, which describes hitherto unknown work that is very close to the user's interest. After discussion with a number of other PAs, in a similar manner to that described in 1. above, the user's PA has already retrieved a relevant technical report from an ftp site, in the anticipation that it will be of interest. A prototype for such a mail filterer is discussed by Maes (1994).
A user is occupied with some task on their computer, when their PA requests their attention: an email message has arrived that contains submission results about a paper they sent to an important conference, and the PA reasoned that they would want to see it as soon as possible. Since the paper has been accepted, the PA begins to look into travel arrangements, by consulting a number of databases and other networked information sources; a short time later, the PA presents the user with a summary of the cheapest and most convenient travel options. Systems such as these are discussed by McGregor (1992) and Levy *et al. (1994).*

A user predicts that they will be unavailable or busy for a certain period, so instructs their PA to deal with their incoming messages in an appropriate manner, as close to his normal behaviour as possible. This could be done by advance specification of his preferences, in addition to the PA using learning techniques (Maes, 1994).

The key computer-based components that appear in each of the above scenarios are known as *software agents*, as opposed to robots (hardware autonomous agents) or human agents (e.g. travel and secret agents). This document presents a brief overview of current agent technology, including consideration of the interface from software agents to other agents. The next section discusses in more detail the denotation of "software agents". The following sections are concerned with the various different categories of agent; background information and discussion about applications of these categories of agent, including commercial applications; agent architectures; agent theories; agent communication languages; and finally, conclusions, with pointers to an on-line summary of agent-related researchers and conferences.

What is an agent?

The original sense of the word "agent" is of one person acting on behalf of someone else. In the field of intelligent interfaces, *software agents* can carry out the user's wishes. Most often, when people in Artificial Intelligence (AI) use the term "agent" they refer to an entity that functions continuously and autonomously in an environment in which other processes take place and other agents exist. They are autonomous in the sense that the agents' activities do not require constant human guidance or intervention. Agents are often taken to be "high-level"; this is manifested in symbolic representation or some cognitive-like function: they may be informable, contain symbolic plans, stimulus-response rules and natural language capabilities[1]. This sense of agents used in the field of artificial intelligence as "high-level" programs that perform tasks for a user can even encompass to a certain extent the notion of agents commonly used in telecommunications, most notably in Telescript[2], meaning mobile script (White, 1994a; White, 1994b). Other types are considered in the next section.

Agents can be as simple as subroutines, but typically they are larger entities with some sort of persistent control (eg. distinct control threads within a single address space, distinct processes on a single machine, or separate processes on different machines). Although the traditional notion of an intelligent agent in AI is that of a rational agent that has explicit representations of its own goals and of its beliefs about the world, the definitions of agent commonly used tend to depend upon the background of the researchers: for example, Genesereth regards software agents solely as "software 'components' that communicate with their peers by exchanging messages in an expressive agent communication language" (Genesereth and Ketchpel, 1994), Shoham regards software agents as ontologically dependant upon their role in a community (Shoham, 1993), and workers in behaviour-

[1] See for example (Walker, 1989).
[2] The programming language written by General Magic Inc. See for example: http://www.genmagic.com/MagicCapDocs/Concepts/Telescript.html.

based artificial intelligence, such as Brooks, employ the notion of a reactive agent, that reacts instinctively to conditions in its immediate environment and does not require an explicit representation of its own goals nor any "world model". Kowalski (1995) attempts to reconcile the conflicting traditional notion of intelligent agent with that of reactive agent.

With respect to the development of user interfaces, a software agent is regarded differently again, as a program that performs tasks for its user, and most definitions also involve some notion of trust (the agent will try to do what it is supposed to do), personalisability (the agent can learn or be explicitly taught what to do for each individual user) and autonomy (the agent is allowed to take at least some actions on the user's behalf, without permission or perhaps even notification).

It is difficult to generalise, but some of the main features of the types of agent used in agent-based software engineering (with the exception of mobile scripts and Brooksean-type agents) are as follows: agents are knowledge based and goal driven; they are able to communicate with each other using a high level language through which knowledge is transmitted and requests are made; and they are capable of action, such as transmission of messages to other agents. Agents that learn, and personal interface agents, are subtypes, as discussed in the following section.

Types of agent

In order to avoid confusion, it might be best if the overused word "agent" were only used if prefixed, and thus its denotation would be clear. Examples of such labels would be planning-agents, smart (ie. intelligent, or reasoning)-agents, ALife-agent, interface-agent, mobile-agent, etc. Such labels need not be exclusive: co-operative agents would apply to agents designed to work as part of a community; autonomous agents might refer to software or hardware robots with conflicting goals or behaviours, etc. The following table distinguishes between various types of agents which are commonly referred to, although there is a great deal of overlap between some of these categories.

Table 1: Various types of Software Agent

Types of agent	Examples	Key Features
Distributed Artificial Intelligence (DAI) agent	task-specific agent	distributed network
	co-operative agent	designed to work as part of a community
	reactive agent	reacts to environment
	Artificial Life (ALife) agent	similar, simple building blocks from whose interactions a complex system may be constructed
	mobile agent	mobility within network

Smart agent	Agent Oriented Programming (AOP) agent	mental state
	planning agent	planning capabilities
	interface agent	assists users in performing computer applications
	learning agent	learning ability
Autonomous agent	softbot with conflicting goals	could be knowledge-based or behaviour-based

These categories shall be considered in more detail in the following sections.

When defining and implementing agents for use in a practical setting, emphasis is placed on a reasoning capability and standardised communication with other agents, in order to facilitate high-level behaviour and ease of extensionality. Such extensionality is two-fold: the modularity inherent in agent architectures provides a long-term benefit of facilitating the augmentation of new functionality, via additional agents; and the standardisation of language enables separate systems to be conjoined. This leads us to the question of whether there are other elements which *must* be included in an agent-based system.

Increasingly, two notions of agency may be seen to have emerged: a *weak notion*, used primarily by the software agents community, particularly for network representation or in reactive systems, and a *strong notion*, used especially within Artificial Intelligence (cf. (Wooldridge and Jennings, 1995a)). The weak notion defines an agent as a self-contained problem solving entity which exhibits the properties of autonomy (operation without the direct intervention of others, and having some kind of control over their actions and internal state), social ability (allowing interaction with other agents, and possibly humans), responsiveness (perception of their environment and timely reaction to changes in this) and proactiveness (taking the initiative, not just exhibiting a stimulus-response type behaviour). The scenarios introduced at the start of this paper provide a good illustration of such properties. The stronger notion of agency augments these properties with conceptualisation or implementation using concepts usually applied to people, such as mentalistic notions, rationality, veracity and learning. The concept of an agent as having mental state to some is a central notion of agency (Shoham, 1993), but this is not a necessary one: even from an Artificial Intelligence standpoint, it may be argued that an agent is just part of a distributed system in which automatic search performs a significant role (and possibly which communicates with its peers in some higher-order language).

The next sections provide background information and discuss work closely related to agency in distributed systems, artificial intelligence and intelligent user interfaces.

Distributed Artificial Intelligence

Research in distributed artificial intelligence concentrates on understanding the knowledge and reasoning techniques needed for intelligent coordination, and on embodying and

evaluating this understanding in computer systems. Such an approach involves compartmentalising problem solving into smaller, more manageable components which communicate with each other. This can result in increased generality, reusability and power. The wider notion of distributed artificial intelligence includes parallel processing, and the whole subject is still in its infancy (Griffiths and Purohit, 1991). A recent survey of distributed artificial intelligence may be found in (Chaib-Draa *et al.*, 1992); a survey of co-operative problem solving techniques may be found in (Corkill and Lesser, 1987).

(a) Agents and Distributed AI

Groups of software agents can make decisions or form coalitions, and hence interest has been generated in agents within distributed artificial intelligence; from such a perspective agents have been regarded as an ideal "foundation of core AI research" (Etzioni, 1993), although more emphasis has traditionally been placed on the overall system behaviour than on its individual subunits.

One application in this field would be the use of agents as information system mediators in order to avoid the user having to supervise a process step by step. For example, a user would not even have to know that a call should be made to a particular destination: they could instead dial a "results" agent, which decides where to look for the information, passes the request to an appropriate information system's agent, which translates the request into a query, so that the message is eventually passed back to the user. A centralised repository of information is not needed. This cascading and subdividing of requests through the cooperation of many agents in order to achieve a goal corresponds to the process described in the first scenario of the introduction above. Other motivations of task-specific distributed AI agents include effective filtering and retrieving information; making more efficient use of resources, and economic applications such as commerical web exploitation (using "payment" of agents, perhaps using a contract net architecture (Davis and Smith, 1983)). Most multi-agent systems are prototypical, with a few exceptions, such as ARCHON - a framework to enable multiple problem solvers to be interconnected so that they can communicate and cooperate with each other whilst solving complex, real-world problems in the field of industrial systems (Jennings, 1991; Jennings *et al.*, 1995); the KAoS architecture used at Boeing (Bradshaw *et al.*, 1996) and Findler and Lo's examination (1988) of distributed planning for air traffic control.

There have been two main differing types of approach within distributed AI, corresponding to the reactive architectures first proposed by Brooks (Brooks, 1986) and now employed in the field of artificial life, versus more traditional symbolic artificial intelligence research involving high-level planning and reasoning.

(b) Behaviour-based v. knowledge-based artificial intelligence

Recently it has become apparent that intelligent behaviour can result without explicit representations or reasoning of the type proposed by symbolic artificial intelligence

(Brooks, 1990); moreover, such behaviour-based artificial intelligence is applicable to agent technology (Maes, 1992).

When agents co-operate in a distributed search problem, they can solve it faster than any agent working in isolation. This is accomplished by having agents exchange hints within a computational ecosystem. Hogg and Huberman (1993) present a quantitative assessment of the value of cooperation for solving constraint satisfaction problems through a series of experiments; one interesting finding is that the bigger a group becomes, the less likely it is that there will be co-operation between members of the group. In the behaviour-based "Enterprise" system the machine sends out a "request for bids" for the task to be done (Malone *et al.* 1988). Other machines are allowed to respond with bids giving estimated completion times which reflect their speed and currently loaded files. The best bid wins, resulting in a flexible and robust system. Similarly, Huberman and Hogg have described a form of distributed computation in which agents have incomplete knowledge and imperfect information on the state of the system, and an implementation (Spawn) based on market mechanisms (Huberman and Hogg, 1991; 1993). However, interaction dynamics among simple components can lead to emergent complexity, and when agents can choose among several resources, the dynamics of the system can be oscillatory and even chaotic (Huberman and Hogg, 1991). Research has been carried out into stabilising such distributed systems (Huberman, 1991). A reward mechanism is desribed for achieving global stability through local controls, whereby the relative number of computation agents following effective strategies is increased at the expense of the others (Waldspurger *et al.*, 1992).

In addition, behaviour-based interface agents have been built (eg. Maes and Kozierok, 1993). Several competence modules are constructed which are experts (or try to become experts) about a small aspect of the task. Oren Etzioni has implemented "softbots", which are intelligent software agents that interact with UNIX by issuing commands and interpreting the environment's feedback (Etzioni, 1994).[3] Otzioni's softbots are able to accept a diverse set of high-level goals, generate and execute plans to achieve these goals in real time, and recover from errors when necessary. The softbot level of discourse in in terms of low-level primitives, and they interact only with the user and not other softbots. Bain and Sammut (1995) describe recent experiments in automatically constructing reactive agents: within the problem field of automatically building controllers for dynamical systems, efficient and robust controllers have been built which reflect human behaviour, are goal-directed and react to the current environment; other general architectures for behaviour-based intelligent autonomous agents are presented by Balkenius (1994) and Malec (1995). This type of learning has been applied in a variety of domains including the control of chemical processes, manufacturing, scheduling and autopiloting of diverse apparatus including aircraft and cranes.

As an example of the differing types of approach of behaviour-based as opposed to knowledge-based AI, knowledge-based AI approaches the problem of building an interface agent in the following way (Sullivan and Tyler, 1991). The agent is given

[3]In general, "bot" is used in reference to software agents. Hence softbot is a software robot, knowbot is a knowledge-based robot, and so on.

knowledge about the problem domain by a knowledge engineer, including a model of the user, a model of the tasks the user engages in, together with a hierarchical specification of the subtasks, knowledge about the vocabulary of these tasks, and so on. At run time, the agent uses this knowledge to recognise the intentions and plans of the user. The problems with this approach are that: it is hard to provide such a complete and consistent model; the model is quickly outdated; because of the computational complexity of the approach, the system would react very slowly; and many kinds of unpredicted events might take place which the agent would not be able to deal with. Instead a behaviour-based interface agent can be built as follows (Maes, 1992; Kozierok and Maes, 1992). Several competence modules are constructed which are experts (or try to become experts) about a small aspect of the task. For example, one module might be responsible for invoking a particular command (like "lpr") at a particular moment. Each of the modules gathers information by observing the user and keeping statistics about a particular aspect of the user's behaviour. From the user's point of view, it will seem as if the system "understands" his intentions, and knows what the problem task involves. Nevertheless, the action sequences are just an emergent property of a distributed system. The idea is that the system will smoothly adapt to the changing habits of the user, will react in a fast way, will never completely break down, and so on.

Meeting scheduling is probably the most popular software agent negotiation application. A traditional, symbolic AI way of solving the problem is described in Kleinrock and Nilsson (1981), using knowledge about scheduling, the particular configuration of the machines and the typical processing jobs at hand. A behaviour-based system could be much more flexible (Maes and Kozierok, 1993).

In summary, distributed artificial intelligence on negotiation and coordination has generally emphasised the need for each agent in a multiagent environment to modify what it does to improve group performance. An alternative way of achieving the latter is to vary the membership of the group, by means of ensuring that agents that perform well in one generation are more likely to propagate to the next (for example by using genetic algorithms), and thus treating the multiagent environment as an ecosystem. Such agents are not necessarily intelligent as individuals, but nevertheless the issues involved are similar to those of mainstream distributed artificial intelligence. In conclusion, both methods suggest an alternative methodology to existing techniques for solving constraint satisfaction problems in computer science.

(c) Mobile Agents

Mobile agents are programs, typically written in a script language, which may be dispatched from a client computer and transported to a remote server computer for execution: advantages offered by mobile agents are assessed by Harrison *et al.* (1995). It could be argued that the programs might as well be executed in the first place, and need not be moved. Certainly, the cost of moving the agents will vary greatly according to the networks, and in some cases could be quite infeasible. General Magic's Telescript system (White, 1994a; 1994b) depends very much on a given infrastructure; Sun's Java on the

other hand is designed for use on the Internet. There are major security issues involved with running programs on other people's systems and the closeness in behaviour to a virus, and in addition such systems will need to take account of promises, commitments, etc. The attractiveness of such systems as agent programming environments is therefore limited; steps are being taken to overcome these issues, such as Safe TCL providing mechansisms for limiting the access provided to a script (for example by placing limits on the number of times a window can be modified by a script), but the safety issues have not yet been fully resolved.

There are many senses of the term "agent" in DAI, and many of these would not be perceived as having individual reasoning capabilities. It is the (non-disjoint!) group of reasoning agents to which the next section directs its attention.

Smart Agents

A great deal of diverse work has been carried out on building intelligent agents for a variety of tasks.

(a) Multi-Agent Planning

First, there has been work in the field of distributed AI and multi-agent planning, as described by Georgeff (1988a; 1988b; 1988c; 1990). This work tackles issues such as co-ordination, synchronisation and control of multiple autonomous agents. In order to predict the future, an agent needs to model other agents as well as to communicate with other agents both to avoid conflicts and to achieve common goals. User interfaces may be enhanced by means of plan recognition, which makes interfaces more intelligent and interactive by providing an intelligent assistant that supports such tasks as advice generation, task completion, context-sensitive responses and error detection and recovery (Goodman and Litman, 1990; 1992). Kautz and Pednault (1988) provide a summary of research in the field of automatic planning and plan recognition, and a formal framework for plan recognition.

(b) Agent-Oriented Programming

There has been a second field of work related to defining models of beliefs, intentions, capabilities, needs, etc. of an agent (Shoham, 1993). Such research is considered in the section on agent theory below.

Agent-oriented programming is a variation of agent-based software engineering, proposed by Shoham (Shoham, 1993). He views agents as entities whose state is viewed as consisting of mental components such as beliefs, capabilities, choices and commitments. Agents communicate with each other through "speech act" type primitives such as informing and requesting, and act on the basis of their mental state. The criterion for any hardware or software component being an agent is whether it has been chosen to be analyzed and controlled in these mental terms: Shoham acknowledges that even a light

bulb *could be* considered to be an agent, although this would not provide a particularly useful abstraction in this case. Even for those accepting this general framework there are variations of approach: for Cohen and Levesque, agents have goals and intentions (Cohen and Levesque, 1990) whereas Shoham regards agents as having obligations and capabilities (Shoham, 1993). Cohen and Levesque (1990) explore the principles governing the balance among an agent's beliefs, goals, actions and intentions: by making explicit the conditions under which an agents can drop his goals the formalism captures a number of important properties of intention.

Exactly how AOP systems should be written is not always clear, but in general a complete AOP system will include three primary components: first, a restricted formal language (used to describe mental state); secondly, an interpreted programming language in which to define and program agents[4], with primitive commands such as REQUEST and INFORM - the semantics of the programming language will be required to be faithful to the semantics of mental state; and finally, an "agentifier", converting neutral devices into programmable agents.

Agent-oriented programming can be viewed as a specialisation *of object-oriented programming*. Like an "object", an agent provides a message-based interface independent of its internal data structures and algorithms. The primary difference between the two approaches lies in the language of the interface, which for AOP is standardised and includes primitives for describing mental state: a computation consisting of the agents informing, requesting, offering, accepting, rejecting, competing, and assisting one another. Table 2, reproduced from (Shoham, 1993), summarises such differences.

[4]This corresponds to the language AGENT0, discussed in the penultimate section.

Table 2: Differences between Object-Oriented Programmed and Agent-Oriented Programming

	OOP	**AOP**
Basic unit	*object*	*agent*
Parameters defining state of basic unit	*unconstrained*	*beliefs, commitments, capabilities, choices, ...*
Process of computation	*message passing and response methods*	*message passing and response methods*
Types of message	*unconstrained*	*inform, request, offer, promise, decline, ...*
Constraints on methods	*none*	*honesty, consistency, ...*

(c) Intelligent User Interfaces

An intelligent user interface is one that exploits knowledge about the dialogue context, the user, and the application system in ways that help the user accomplish his interaction goal efficiently. Such interfaces may encompass intelligent front ends, natural language interfaces, tutoring systems, intelligent help and support systems, multimedia presentation systems, dialogue assistants, etc.: see Hefley and Murray (1993) for an overview of the subject.

An intelligent interface cannot just respond to its user's commands and queries - it must also be able to take the initiative in order to volunteer information, correct user misconceptions or reject unethical user requests. Intelligent agents, which are cooperative agents acting as an intermediary between users and application systems, are needed to perform these functions, and this area of research is expanding rapidly (Kay, 1994). Agents can operate in conjunction with users' existing applications, helping them perform tedious, repetitive or time-consuming tasks more easily and efficiently (Palaniappan *et al.*, 1992). Early work addressing this problem carried out by Waterman produced the Rule-directed Interactive Transaction Agent system, used for developing interface agents organised as sets of IF-THEN rules; there are agents which interface the user to computer systems he wishes to use, and others which interact with the user to acquire the knowledge needed to create these interface programs (Waterman, 1978).

Existing machine techniques for acquiring user models can be characterised along five orthogonal dimensions: passive/active, user-initiated/automatic, logical/plausible, direct/indirect and on-line/off-line (Chin, 1993). Frohlich (1993) reviews developments in the implementation and understanding of direct manipulation interfaces.

Various intelligent interfaces have been built, among which the following are of particular interest to agency:

- Kaye and Karam (1987) describe an implemented network of distributed cooperating knowledge-based or expert assistants and servers, intended for high-level support of office workers. The system is capable of supporting concurrent

multiple consultations or tasks and has facilities for the interruption and resumption of consultations as appropriate.

- The XTRA system is an example of an intelligent multimodal interface which offers a combination of graphics and natural language for input and output (Wahlster, 1989). The generation of a multimodal presentation can be considered as an incremental planning process that aims to achieve a given communicative goal - plan-based integration of natural language and graphics generation is considered in Wahlster *et al.*, (1993).
- Chin (1991) describes UCEgo, the intelligent agent component of UC, a natural language interface that helps the user solve problems when using the UNIX operating system (Wilensky *et al.*, 1988).
- Object Lens is a general tool for supporting many kinds of cooperative work and information management applications, with particular emphasis on supporting tailorability, integration and semiformality (Lai and Malone, 1991). Details of how such a tool may be tailored to provide a variety of integrated information management and collaboration applications are described in Malone *et al.* (1992).
- Cypher (1993) describes a system (namely, Eager) which uses programming by example to antipate a user's actions and offer to complete tasks; it uses a new interface technique which highlights menus and objects on the screen to indicate what it expects the user to do next, whilst interfering minimally with the user's normal activities.
- The Dialogue Modelling Project at Hewlett Packard Labs, Bristol has carried out research into providing a multimodal interface for a real sales database application (Haddock, 1992).

Interface Agents

Work on intelligent interface agents is motivated by defining interface agents as computer programs that employ Artificial Intelligence techniques in order to provide assistance to a user dealing with a particular computer application (Maes and Kozierok, 1993).

Much work on interface agents is being carried out by the computer-supported cooperative work community. Interface agents can communicate with a network of distributed AI-type agents, or themselves form a network. They may be associated with users, in order to allow customisation. Thus, agents may be used as personal mediators: each user having his own personal agent that receives messages, and prioritises them, automatically acts upon them independently of the user, etc. For example, an agent could note a request for a meeting, place that meeting in its diary, and confirm the time of the meeting to the requester. When the user needed help, he could contact his personal agent. The agent would try to satisfy any resulting goals using the distributed problem-solving model. To date, interface agents have tended to be highly elaborate, custom-crafted programs designed for very specific applications, as opposed to the more generic nature of DAI agents. Motivations for the development of interface agents include the current trend towards less experienced computer users, thus enhancing the benefit of hiding complexity (for example there could be one command for multiple tasks), performing tasks on the

user's behalf, teaching the user (eg. using an expert's agent), helping different users collaborate, monitoring events and procedures, etc.

One of the most challenging aspects of agent design is to define specific tasks that are both feasible using current technology, and are truly useful to the everyday user. Assisting in education and training has been a worthwhile aspect of agent research over the past decade, but the interface agents being developed today cross a number of application boundaries. They are designed to filter and gather information from commercial data services and public domains like the Internet and to automate work flow: agents have been used to assist users in a range of daily activities, such as setting up meetings, obtaining reports, locating information in multiple databases, finding the person fulfilling a given role, tracking the whereabouts of people, and so on. The agents should blend transparently into normal work environments, while relieving users of low-level administrative and clerical tasks. On the practical side, users should feel that the agents are reliable and predictable, and that the human user remains in ultimate control. The key feature of current research is the development of personalised agents that mediate communication between users and task-specific agents (possibly, ultimately, including software agents developed externally). Issues include the separation of task-specific agents from user-centred agents, the need to handle issues of security and privacy, the need for good human interfaces and high reliability. In addition, there is another dimension of interest, for in each of these categories a subset of researchers have been developing agents which apply learning techniques to discover such things about the user as his level of expertise, his programming style, or his areas of personal and professional interest (see following subsection).

Assistant agents could be one of several types. They could be advisory, task-assisting, act as proxy or be concerned with information filtering or discovery. Advisory agents offer instruction and advice to the user to help him carry out his work (applications include tutoring systems). Ted Selker's COACH program is a well-known example of a LISP tutor which tailored appropriate "help" feedback to the user via analysis of their input (Selker, 1994). Research on student modelling in such intelligent tutoring systems has grown out of this area: of note is John Anderson's work (Anderson *et al.*, 1990), and extensions to modelling agents in real-time, dynamic environments (Tambe, 1995). User-agent relationships can be viewed as being similar to a manager-secretary relationship, and so agents are also being used to automate multiple-tool tasks (Toomey and Johnson, 1994). Such task-assisting agents are participative agents, or "prescient agents" (McGregor, 1992). As opposed to advisory agents, assistant agents will not necessarily provide user feedback. Semi-autonomous, high-level software agents can be associated with each individual user, as well as with other components of the system.[5]

Kahn and Cerf proposed an architecture for a set of information-management agents, called "Knowbots" (Kahn and Cerf, 1988). The various agents are specifically designed and built to perform particular tasks. Etzioni *et al.* have built agents for the Unix domain

[5]Research has been carried out at Hewlett-Packard to develop cooperative interfaces to information management systems (Stenton, 1990) and agents that act as a personal assistant for the user in the NewWave environment (Stearns, 1989; Bedford-Roberts, 1992).

that can perform a variety of Unix tasks (Etzioni *et al*., 1992; 1994) . This work has focused extensively on reasoning and planning with incomplete information, which arises in many of these tasks. Software agents prototypes have also been built to assist users in scheduling meetings and managing their personal calendars (Maes and Kozierok, 1993; Jennings and Jackson, 1995), typical activities in a university-based research project (the OFFICE system) (Nirenburg and Lesser, 1988) and for note taking (Schlimmer and Hermens, 1994). In addition agents may be created as a human proxy, by managing incoming calls and automatically providing services, which is of particular benefit when the user is busy or unavailable. Some of the implications associated with presenting and programming intelligent assistant agents are considered by Whittaker (1990) and Nass *et al. (1994; 1995).*

Many applications of agents include information filtering and information discovery. Software agents have already been used to retrieve documents from disparate databases (Vorhees, 1994), and to filter information on behalf of their user (Levy *et al.,* 1994). Knoblock and Arens (1993) focus on flexible and efficient retrieval of information from heterogeneous information sources, while Rus and Subramanian (1994) have developed a bibliography agent. Academic research is currently being carried out into applications using agency which are able to deal in an intelligent manner with email, as discussed for example by McGregor (1992). There has been an increased interest in collaborative filtering (namely, people collaborating to help one another perform filtering by recording their reactions to documents they read) (Goldberg *et al*., 1992). Various types of information manager that filter and obtain information on behalf of their users have been prototyped (Maes, 1994). Prototypes have been built for email handling (MAXIMS), meeting scheduling, electronic news filtering (only using keywords) (NEWT), selection of entertainment (but needing virtual users!) (Maes, 1994), and intelligent information-sharing systems (Malone *et al*., 1987). Research is also being encouraged in this field in Japan: the FRIEND21 ("Future Personalised Information Environment Development") project is a Japanese national project for developing interface architecture (using the "Agency Model") for computer machinery for the 21st century information environment (Nonogaki and Ueda, 1991).

A common recent application is the development of internet agents, since agents provide an appropriate tool for managing the vast amounts of available information on the World Wide Web. Some Internet agents attempt to present an integrated view of the Internet as a whole, but the most common to date are information gatherers, which traverse the web and then report what they find to a home location. WWW agents include JumpStation, WebCrawler and SIFT, Stanford University's personalised Net information filtering service: see Martin Koster's web pages for a current list (http://web.nexor.co.uk/mak/doc/robots/robots.html). Research papers on intelligent user agents for information filtering include: a system which filters news articles on the Internet (McElligott and Sorensen, 1994); agents used to adapt the structure of a hypertext according to the behaviour and preferences exhibited by its users while browsing (Stotts and Furuta, 1991); and building agents for retrieving information from the Internet, with a focus on efficient query processing (Levy *et al.,1994).*

(d) Learning Agents

In addition to reactive systems which learn from the environment, which have already been considered in an earlier section, there has been a growing trend within machine learning to investigate learning methods that involve interacting with other agents, and more specifically, learning by instruction from the user (Lieberman, 1993; Gordon and Subramanian, 1993; Maclin and Shavlik, 1994); learning from observation (Payne *et al.*, 1995); and knowledge acquisition and refinement (Davies and Edwards, 1995; Byrne and Edwards, 1995).

Although many of the issues are of relevance across the whole field of software agent applications, learning agents are especially associated with intelligent interfaces. Such machine learning systems include MINDS, a distributed system of knowledge-based query engines in which document distribution patterns, as well as user interests and preferences during system usage, are learned in order to customize multimedia document retrievals in the office domain (Mukhopadhyay *et al.*, 1988). Agents that learn the user's needs over time build a model of their users' intent and then assist as appropriate. Similarly, Schlimmer and Hermens (1993) have built a learning-apprentice interactive note-taking system for pen-based computers which predicts the user's input and automatically constructs a user interface on request using learning techniques. Dent *et al.* (1992) and Maes and Kozierok (1993) describe the design of interface agents that learn how to assist users in scheduling meetings and managing their personal calendars, and were implemented using memory-based learning and reinforcement learning techniques. The agents act as personal assistants collaborating with the user, and learn the user's interests, habits and preferences, although the user has overall control. Interface agents can learn by means of observing and imitating the user, user feedback (both directly requested and unsolicited), the user deliberately programming by example, and in addition by asking other agents (cf. (Maes, 1994)). Such an approach to building interface agents is feasible and has several advantages over other approaches: it provides a customised and adaptive solution which is less costly and can ensure better user acceptability.

(e) Believable Agents

In this area researchers are concerned with applying agent technology to create characters in virtual environments such as animation: such characters are not so much intelligent as versatile. The area is still very young: Joseph Bates' OZ project, in which artificial characters are developed which allow the "suspension of disbelief", is probably the most well known at present (Bates, 1994).

(f) Commercial Systems

Current commercial agents that assist users are designed only for specific domains, and include Smart mailboxes and search engines (Reinhardt, 1994). "Edify" (Santa Clara, CA) offers an agent-supporting environment, called Electronic Workforce, which automates

human-resources queries and customer-service telephone support and deals with the underlying paperwork of an employee-review process. IBM has various projects (see (Aparicio *et al., 1995) and* http://activist.gpl.ibm.com:81) including the "Communications System", an intelligent new network incorporating smart mailboxes, due to be released at the present time (Lawrence, 1995). BT (Martlesham) is also working on intelligent routing technology, for future use in network management. (Guilfoyle, 1995) provides an overview of organisations with active current activities in agent technology, plus the commercial effects of this field. Nearly forty organisations are quoted as being active in intelligent agents, and this list is not exhaustive: a summary of the findings are reproduced below in Table 3.

Table 3: A Selection of Active Organisations within Agent Technology

Organisation	***Intelligent agent activities***
Alcatel	uses intelligent agent design model in telecoms equipment to improve management capabilities
Andersen Consulting	Commander Exception Monitor agent-based data mining tool, developed with ComShare
Apple	uses agents in Newton PDA, agent-like querying via HyperCard
AT&T	includes PersonaLink service, using Telescript
Bellcore	research work including message management, agents in information retrieval
Beyond Incorporated	BeyondMail, PowerRules use intelligent task automation for email filtering, routing
British Telecom	R&D work on advanced multi-agent systems; expects to use in network management in future
Carnegie Group Inc.	uses agent design model in custom work eg. with US West
Charles River Analytics	OpenSesame! learning interface product; also licensing technology for embedded intelligence
CIA Software	introduced Competitive Intelligence Agent personal assistant to find and access information sources
Computer Sciences Corporation	interested in developing software based in intelligent agents
D&B Software	intelligent agent feature in SmartStream workflow
Daimler-Benz	using agent-oriented techniques for various applications
Delphi Internet Services	developing intelligent agents to help users find information on the Internet
Digital Equipment Corp.	agent work including Polycenter Path Doctor for network management
Dow Jones	working on agent-based services
ECRC	research in inter-agent languages
Edify	supplies Electronic Workforce agent environment for customer service support, etc.

General Magic	develops technologies for Telescript scripting language and Magic CAP PDA interface for agent-based communications(see http: //www.genmagic.com)
Hewlett-Packard	includes NewWave agents, network management agents to support filtering, routing and higher-level tasks, e.g. software distribution
IBM	various projects including Intelligent Communications messaging hub (see http://www.ibm.com/technology)
ICON	AI software house developing applications including agent-based credit management
Logica	develops custom systems, intelligent agent development environments
Lotus	simple agent functions; Notes 4 adds to agent element(see http://www.lotus.com)
MIT	several projects on intelligent agents, inclusing task automation and user interface work
Microsoft	IntelliSense macro type feature in several products; "Bob" user agent for residential market demonstrated
No-Hands	sells Magnet agent for task automation in file moving
Oracle	Oracle in Motion database client-server offering uses intelligent agents to support wireless infrastructure; ConText linguistic analysis tool to help cope with information retrieved
Pilot software	EIS/workflow supplier reportedly developing agent add-on to its product
Quasar Knowledge Systems	sells Smalltalk Agents enhanced version of Smalltalk
Reuters	working on agent-based services
Sandpoint	Hoover agent-based search program
Sharp Laboratories of Europe	research work on simple communicative agents
Sun Microsystems	plans to distribute Tcl agent scripting language
Synoptics	agent-based network management products
US West	using applications containing intelligent agents
Verity	developed Topic text retrieval and InfoAgent Developer Kit; partners include Lotus, Adobe
WildSoft Inc.	working with wireless communications, client-based software to filter and forward email

Telescript (White, 1994a;1994b) is the first commercial agent language. Telescript technology is the name given by General Magic to a family of concepts and techniques which they have developed to underpin their products. The two key concepts are of places: virtual locations that are occupied by agents (corresponding to a single machine, or several machines), and agents: the providers and consumers of goods in the electronic marketplace applications that Telescript was developed to support. Telescript agents have an associated permit, which specifies what the agent can do, and what resources it can use. The most important resources are "money", measured in "teleclicks" (which correspond to real money), lifetime (in seconds), and size (in bytes). Engines (agent

operating systems) continually monitor agent's resource consumption, and kill agents that exceed their limit. They also provide (C/C++) links to other applications via application program interfaces (APIs). Agents and places are programmed using the Telescript language (see below), and General Magic claim that the sophisticated built-in communications services make Telescript ideal for agent applications. So far only AT&T has developed a commercial service based on Telescript (the "PersonaLink" service), but other companies are interested. This technology is likely to have a significant impact, particularly because it has received support from Apple, AT&T, Motorola, Philips and Sony, and now, perhaps in response to a version of Sun's Java being made freely available, the Telescript development environment can be downloaded from http://www.genmagic.com/.

Intelligent agents are potentially very important for suppliers. Commercial advantages of agents include the augmentation of traditional support applications with automated, round the clock services. Although the field is relatively young, the potential for commercial success is encapsulated by the following quote from Irene Greif, Director of Workgroup Technologies at Lotus (Greif, 1994):

> *"... we believe that agents-centred design can potentially revolutionise UIs and produce usefulness. We also believe they can be built from current technologies, used in new ways and architected to anticipate future advanced in process representation, AI and natural language understanding. We predict enormous benefits from agents delivered in product well before anyone can master and bring to product many of the more ambitious aspects of intelligent agents."*

This section has reviewed agency-related research in artificial intelligence, including agent-oriented programming and intelligent user interfaces. The following sections review architecture of multi-agent systems, agent theories and agent communication languages.

Architecture of Multi-Agent Systems

The variants of multi-agent systems' architectures reflect differing approaches, such as the artificial life approach of simple agents communicating directly in a network, or systems which involve symbolic reasoning and planning.

In general, agents can communicate by either direct or assisted communication. Direct communication takes the form of a contract-net type architecture or specification sharing. In the former, agents in need of services distribute requests for proposals to other agents. The recipients of these messages evaluate those requests and submit bids to the originating agents. The originators use these bids to decide which agents to task and then award contracts to those agents (Davis and Smith, 1983; Smith, 1980); this relates to the behaviour-based research discussed in a previous section, and specifically to Malone and Huberman's work (Malone *et al.*, 1988; Huberman, 1993). Negotiation can be used to achieve different goals, such as distributing control and data to avoid bottlenecks and enabling a finer degree of control in making resource allocation and focus decisions than is

possible with traditional mechanisms. Agent-oriented programming shares with early work on "contract nets" a notion of contracts between agents. However, contract nets are based on broadcasting contracts and soliciting bids, and the contract net approach has no other notion of mental state, no range of communicative speech acts, nor the asynchronous, real-time design inherent in agent-oriented programming (Shoham, 1993). There has also been much theoretical work on abstract agent negotiation protocols (Zlotkin and Rosenschein, 1994).

In specification sharing, agents supply other agents with information about their capabilities and needs; and these agents can then use this information to coordinate their activities - this is often more efficient than a contract-net type architecture. Assisted coordination is when agents rely on special system programs to achieve coordination, resulting in a federated system. Agents communicate only with system programs called facilitators, and the facilitators communicate with each other. Agents document their needs and abilities for their local facilitators. In addition, they send (and accept) application-level information and requests to/from their facilitators. Facilitators use the documentation provided by these agents to transform these application-level messages and route them to the appropriate places (Genesereth and Ketchpel, 1994). This idea is a development of the notion long current in distributed systems of a trader, which may be used to facilitate communication between agents, without prohibiting direct communication between the agents themselves.

Many variants of such architectures have been proposed: for example, Palaniappan *et al.*, (1992) proposes a distributed, open architecture for computer-based agents that operate in conjunction with users' existing applications in the electronic desktop. Knoblock and Arens (1994) have developed their own architcture for information retrieval agents and have built agents that plan and learn in the transportation planning domain, in order to retrieve information from heterogeneous sources, having located it. These agents contain a detailed model of this domain and extract information from a set of nine relational databases. The agents select appropriate information sources, generate parallel plans, execute the queries in parallel, and learn about the information sources. In such situations agents have the advantage of potentially acting as a broker, or building up appropriate levels of confidence about differing sources of information. The application of a formal specification framework, originally developed for complex reasoning systems, to multi-agent systems, is described by Brazier *et al.* (1995a;1995b). An agent architecture for distributed medical care, which may also be used in more general support applications, is presented by Huang *et al.* (1994; 1995a; 1995b).

Note that there is also a major research topic concerning how agents can be decomposed into sets of component modules and how these modules should interact: such (internal) agent architectures may be categorised into reasoning (the prevalent approach, which uses symbolic modelling and reasoning), reactive (i.e. Brooksean, which does not use such explicit representations and intelligence is an emergent property of the system) and hybrid architectures. Examples of each type of architecture would be the Intelligent Resource-bounded Machine Architecture (IRMA) (Bratman *et al.*, 1988) (upon which an air traffic

control system in Sydney airport was based), the subsumption architecture (Brooks, 1986) and theTouringMachines Architecture, which has a reactive, a planning and a modelling layer (Ferguson, 1992), respectively. The proceedings of ATAL-94 (Wooldridge and Jennings, 1995b) contains a separate section on agent architectures, and Pattie Maes (1990) has edited a book describing different reactive agent architectures. Further details are also discussed by Wooldridge and Jennings (1995a).

Agent Theories

Agent theories are formal accounts of agent behaviour and structure, providing a semantics for agent architectures, languages and tools. The philosopher Daniel Dennett has coined the term intentional system to describe entities "whose behaviour can be predicted by the method of attributing belief, desires and rational acumen" (Dennett, 1987, P243) and viewing agents as intentional systems can provide a useful abstraction tool with which to analyse and explain complex systems. Cohen and Levesque's work in developing a theory of intention has formed a basis for much further work (1990;1995), as has Rao and Georgeff's belief-desire-intention (BDI) model (1995). Agent formalisms are often intended as specification languages (Wooldridge and Jennings, 1995). Besides logics of knowledge and belief, which the logic of mental state extends, relevant work includes the Intelligent Communicating Agents project (1987-88), Rosenschein and Kaelbling's situated automata (Rosenschein and Kaelbling, 1986) and Genesereth's research on informable agents (Genesereth, 1991).[6] For an overview of related research see Shoham (1993).

Agent Communication Languages

In agent-based software engineering, agents use a common language with an agent-independent semantics. The Darpa Knowledge Sharing Effort (Neches *et al.,* 1991) has encouraged much agent-based research into knowledge representation and communication languages. This effort has led to the design of an *agent communication language* (ACL), which is intended as a standardised agent communication language. Michael Genesereth at Stanford's Logic Group has presented a "federation" agent architecture that employs ACL, a necessary and sufficient condition of being an agent in their view being its ability to communicate with other agents via a given language (Genesereth and Ketchpel, 1994).

ACL consists of three parts. First, there is the vocabulary, which are "words" from an English database. Secondly, there is the inner language KIF (Knowledge Interchange Format), which corresponds to terms and sentences of logic: KIF is an enhanced version of the language of first-order predicate calculus, which is used to declare information, information about information, and procedures. Finally, there is a linguistic layer in which context is taken into account, corresponding to logical expressions, namely KQML, standing for knowledge query and manipulation language (Finin *et al*., 1992). Each KQML message consists of a communications type and one or more KIF expressions.

[6]Kave Eshghi, at HP Labs Bristol, has worked on Abductive Oriented Programming (an extension of AOP), and uses a truth maintenance system to keep track of the mental state of an agent (Eshghi, 1993).

The communications type might be a simple query or command, a request for bids or a delayed request. Complete specifications of KIF and KQML, as well as cource code for much of the Logic Group's work is available on the WWW at http://logic.stanford.edu. However,
this language is extremely rich, and a subset of this language would suffice for most applications
it is likely that modifications of current systems will be used, and it is most unlikely that existing ('legacy') software will be rewritten to fit any given standard (alternatives are to use a transducer (a translator) or a wrapper).

Alternative agent communication languages have been developed. These can be divided into two classifications: weak and strong languages, corresponding to the weak and strong notions of agent definition. Weak languages would include the ACTOR languages, TCL/TK and Telescript; strong languages would include AGENT0, SodaBot and Concurrent MetateM.

Hewitt's ACTOR language was developed in the late 70s and early 80s with his co-worker Agha, and laid the foundations for early development of concurrent object-based languages (Agha, 1986). ACTORS are similar to, but weaker than, agents: they are not really autonomous or pro-active, but are social, reactive and message-driven.

The Tool Control Language (TCL) is a standard command language, which is interpreted, extendable and can be embedded in an application, and is associated with TK, an X-window based widget toolkit which provides facilities for making GUI features and for interprocess communication, via the exchange of TCL programs (which are called *scripts)*. TCL scripts have many of the properties of UNIX shell scripts, such as being plain text programs, being able to be executed by a shell program and call up and obtain results from other programs. TCL scripts can be regarded as being agents in applications where the scripts are exchanged across a network, and executed on remote machines. However, although the core primitives may be used for building agent programming environments, this was not the intention for use of the language.

Telescript is an object-oriented language-based environment (apparently based on Smalltalk) for constructing multi-agent systems, developed by General Magic, Inc. (White, 1994a;1994b). The language is interpreted, persistent, a "process" class (with "agent" and "place" as sub-classes), and has two levels: the "visible" language, and a semi-compiled language for efficient execution. Agents are interpreted programs, rather like TCL. They are mobile, and can communicate with each other across a network, if occupying different places, or by meeting at the same location (where they can share resources such as teleclicks).

Other weak languages include April, a language for building agent applications that supports multi-tasking, network transparent message-passing, and symbolic processing, which was developed at Imperial College as part of the ESPRIT project IMAGINE

(McCabe and Clark, 1995); AgenTalk, a co-ordination protocol description language for multiagent systems (Kuwabara *et al., 1995), and many more.*[7]

The most famous example, besides ACL, of a strong agent communication language would probably be Shoham's prototype AGENT0, an interpreted programming language for programming agents developed to illustrate the principles of agent-oriented programming (AOP). It is implemented in LISP, with each agent having four component sets: capabilities, initial beliefs, initial commitments, and commitment rules. Each commitment rule contains a message condition, a mental condition and an action. If the first two conditions are met, then the rule fires and the action becomes added to the agent's commitment set. Thomas developed this prototype to include a planning capability and facility for communication of requests for action via high-level goals, again using mental change rules, resulting in her Planning Communicating Agents language, PLACA (Thomas, 1995). Other strong agent languages include Concurrent MetateM - a multi-agent language in which each agent is given a temporal logic specification of the behaviour it should exhibit (Fisher, 1994) - and SodaBot, a general-purpose software agent user-environment and construction system, with a new language for programming the basic software agents, where the agents' primitives are designed around human-level descriptions of agent activity (Coen, 1994).

Conclusions

There is no clear and universal notion of agent, and definitions depend very much on researchers' interests. Many of the software agent community are trying to develop scripting languages and transport protocols which make it easy for programs ("agents") in a distributed system to communicate and move around. This is suitable for such applications as loading short programs over the world-wide web, but not so much for many of the other actions proposed, such as querying remote databases, etc. In the latter case the agent has to know exactly in what format the remote data is stored (and in addition what the data means), will consume resources at the host end (at the expense of other users), will not be optimised for querying the remote database (and is unoptimisable by the host due to the opaque nature of scripting languages), raises security issues, and so on. Moreover, the more complicated the software agent, the larger it will be and hence the longer it will take to transmit. Instead, it seems more sensible to make the host ends start agents in their own right and to communicate requests at a higher level, namely by stating what they want rather than how to get it. In this framework, only the recipients of a request need have intimate knowledge of the data and services it provides. Requests can be translated into queries optimised for the local environment and can take advantage of internal structures not visible to the outside world. To do all this requires some translation scheme and a reasoning engine. Since agents need to ask for services and information from one another, and these requests are generated in plans, agents need to be able to reason about knowledge they will have when they start executing a plan, but do not have at the time of plan generation (a simple example being planning to call the operator to find out

[7] An extensive list and further details of agent communication languages may be found in the Distributed Artificial Intelligence and Multi-Agent Systems Archive, via web site: http://www.info.unicaen.fr/~serge/sma.html.

someone's number, before calling them). However, for networks which undergo frequent change, such planning would not work very well, and it makes sense to spread the management programs across the network rather than localising them in a few centres (Lawrence, 1995), or to use reactive systems.

While today's agent-oriented research indicates the technology's potential, and various prototype systems have been built, it will take more development work to produce wide-scale commercial applications. An interagent communications language will help overcome one of the technical hurdles by reducing the time and efforts developers need to build an agent, although it is likely that this will develop piecemeal rather than by the initial introduction of a common standard. In addition, significant social issues, such as agent security and responsibility for an agent's actions, need to be resolved. However, an excellent opportunity is available for bridging such gaps in order to exploit agent technology and thereby produce practical systems of substantial commercial value.

For details of key agent-related groups, researchers and conferences, including hypertext links, see Viceroy WWW pages: http://www-incl.hpl.hp.com/projects/agency/Agency.html; http://www-uk.hpl.hp.com/projects/Viceroy.html.

Acknowledgements
I would like to thank Chris Preist, David Frohlich and Janet Bruten of HP Labs, Bristol, and Ralph Becket of Cambridge University, for helpful feedback, discussion and information.

References

G. Agha. ACTORS: A Model of Concurrent Computation in Distributed Systems. MIT Press. 1986.

J.R. Anderson, C.F. Boyle, A.T. Cobett and M.W. Lewis. Cognitive modelling and intelligent tutoring.
Artificial Intelligence, 42, 7-49, 1990.

M. Aparicio *et al. IBM Applications of Intelligent Agents.*
In *Intelligent Agents and their Business Applications, Unicom, Uxbridge, 197-204,* November 1995.

C. Balkenius. Natural Intelligence for Autonomous Agents.
Lund University Cognitive Studies, 29, 1994.

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

J. Bedford-Roberts. The Personal Filing Assistant.
Internal Report, Hewlett-Packard Laboratories, Bristol, June 1992.

J.M. Bradshaw, S. Dutfield, P. Benoit and J.D. Woolley. KAoS: Toward an Industrial-Strength Open Agent Architecture.
In *Software Agents, ed. J.M. Bradshaw, AAI/MIT Press, 1996.*

M.E. Bratman, D.J. Israel and M.E. Pollack. Plans and resource-bounded practical reasoning.
Computational Intelligence, 4, 349-355, 1988.

F. Brazier, B.D. Keplicz, N.R. Jennings and J. Treur. Formal Specification of Multi-Agent Systems: a Real-World Case.
First International Conference on Multi-Agent Systems, San Francisco, CA, 25-32, June 1995.

F. Brazier, B.D. Keplicz, N.R. Jennings and J. Treur. Modelling Distributed Industrial Processes in a Multi-Agent Framework.
In *Towards the Intelligent Organisation: The Coordination Perspective, eds. S. Kim and G.M.P. O'Hare, Springer-Verlag, 1995.*

R.A. Brooks. A robust layered control system for a mobile robot.
IEEE Journal of Robotics and Automation, 2:1, 14-23, 1986.

R.A. Brooks. Elephants don't play chess.
In *Designing Autonomous Agents, ed. Maes, The MIT Press, Cambridge, 3-15, 1990.*

C. Byrne and P. Edwards. Refinement in Agent Groups.
In *Proceedings of IJCAI'95 Workshop on Adaptation and Learning in Multi-Agent Systems, Montreal, Canada, August 1995. (To be published by Springer-Verlag as part of the Lecture Notes in AI Series).*

B. Chaib-Draa, B. Moulin, R. Mandiae and P. Millot. Trends in distributed artificial intelligence.
Artificial Intelligence Review, 1:6, 35-66, 1992.

D.N. Chin. Intelligent interfaces as agents.
In *Intelligent User Interfaces, eds. J.W. Sullivan and S.W. Tyler, ACM, New York, 177-206, 1991.*

D.N. Chin. Acquiring user models.
Artificial Intelligence Review, 7:3, 185-197, August 1993.

M.H. Coen. SodaBot: A Software Agent Environment and Construction System.
MSc, AI Technical Report No. 1493, MIT AI Lab, June 1994.

P.R. Cohen and H.J. Levesque. Intention is choice with commitment.

In *Artificial Intelligence,* 42:3, 213-261, 1990.

P.R. Cohen and H.J. Levesque. Communicative actions for artificial agents.
In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, CA, 65-72, June 1995.*

D.D. Corkill and V.R. Lesser. Distributed problem solving.
In *Encyclopaedia of Artificial Intelligence, ed. S.C. Shapiro, John Wiley and Sons, New York, 245-251, 1987.*

A. Cypher. Eager: Programming repetitive tasks by example.
In *Watch What I do: Programming by Demonstration, eds. A. Cypher et al., MIT Press, Cambridge, MA, 1993.*

W. Davies and P. Edwards. Distributed Learning: An Agent-Based Approach to Data-Mining.
In *Proceedings of ML'95 Workshop on Agents that Learn from other Agents, Tahoe City, California, July 1995.*

R. Davis and R.G. Smith. Negotiation as a Metaphor for Distributed Problem Solving.
In *Artificial Intelligence,* 20:1, 63-109, 1993.

D.C. Dennett. *The Intentional Stance.*
MIT Press, 1987.

L. Dent, J. Boticario, J. McDermott, and D. Zabowski. A personal learning apprentice.
In *Proceedings of AAAI-92, AAAI Press/The MIT Press, 96-103, 1992.*

K. Eshghi. Using Software Agents for Facilitating Access to On-Line Resources.
HPL-93-82. September, 1993.

O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh and M. Williamson. An approach to planning with incomplete information.
In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA, 1992.*

O. Etzioni, N. Lesh and R. Segal. Building Softbots for UNIX (Preliminary Report).
In *Software Agents - Papers from the 1994 Spring Symposium, ed. O. Etzioni, AAAI Press, 9-16, 1994.*

I.A. Ferguson. Towards an architecture for adaptive, rational, mobile agents.
In *Decentralized AI 3 - Proceedings of the Third European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-91), ed. E. Werner and Y. Demazeau, Elsevier, 249-262, 1992.*

N.V. Findler and R. Lo. An Examination of Distributed Planning in the World of Air Traffic Control.
In *Readings in Distributed Artificial Intelligence, ed. A. Bond and L. Gasser, Morgan Kaufmann, San Mateo, 617-628, 1988.*

T. Finin, R. Fritzson and D. McKay. A language and protocol to support intelligent agent interoperability.
In *Proceedings of the CE and CALS, Washington DC, June 1992.*

M. Fisher. A Survey of Concurrent MetateM - the language and its applications.
In *Temporal Logic - Proceedings of the First International Conference, ed. D.M. Gabbay and H.J. Ohlbach, Springer-Verlag, 480-505, 1994.*

D.M. Frohlich. The history and future of direct manipulation.
Behaviour and Information Technology, 12:6, 315-329, 1993.

M.R. Genesereth. A comparative analysis of some simple architectures for autonomous agents.
In *Architectures for Intelligence, ed. K. VanLehn, Lawrence Erlbaum, Hillsdale, NJ, 1991.*

M.R. Genesereth and S.P. Ketchpel. Software Agents.
Communications of the ACM, 37:7, 48-53, 1994.

M.P. Georgeff. Communication and Interaction in MultiAgent Planning.
In *Readings in Distributed Artificial Intelligence, ed. A. Bond and L. Gasser, Morgan Kaufmann, San Mateo, 200-204, 1988.*

M.P. Georgeff. A Theory of Action for MultiAgent Planning.
In *Readings in Distributed Artificial Intelligence, ed. A. Bond and L. Gasser, Morgan Kaufmann, San Mateo, 205-209, 1988.*

M.P. Georgeff. The Representation of Events in MultiAgent Domains.
In *Readings in Distributed Artificial Intelligence, ed. A. Bond and L. Gasser, Morgan Kaufmann, San Mateo, 210-215, 1988.*

M.P. Georgeff. Planning.
In *Readings in Planning. Morgan Kaufmann Publishers, San Mateo, CA, 1990.*

D. Goldberg, D. Nichols, B.M. Oki and D. Terry. Using Collaborative Filtering to Weave and Information Tapestry.
Communications of the ACM, 35:12, 61-70, 1992.

B.A. Goodman and D.J. Litman. Plan recognition for intelligent interfaces.

In *Sixth Conference on Artificial Intelligence Applications, IEEE Comput. Soc. Press, 297-303, 1990.*

B.A. Goodman and D.J. Litman. On the interaction between plan recognition and intelligent interfaces.
User Modeling and User-Adapted Interaction, 2:1-2, 83-115, 1992.

D. Gordon and D. Subramanian. A Multistrategy Learning Scheme for Agent Knowledge Acquisition.
Informatica, 17, 331-346, 1993.

I. Greif. Desktop agents in group-enabled products.
Communications of ACM, 37:7, July 1994.

D.G. Griffiths and B.K. Purohit. Fundamentals of distributed artificial intelligence.
*British Telecom Technology Journa*l, 9:3, July 1991.

C. Guilfoyle. Vendors of intelligent agent technologies: a market overview.
In *Intelligent Agents and their Business Applications, Unicom, Uxbridge, 136-142, November 1995.*

N.J. Haddock. Multimodal Database Query.
Proceedings of the 15th International Conference on Computational Linguistics, 1274-1278, 1992.

C.G. Harrison, D.M. Chess and A. Kershenbaum. Mobile Agents: Are they a good idea? Research Report, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, March 1995.

W.E. Hefley and D. Murray. Intelligent User Interfaces.
Proceedings of the 1993 International Workshop on Intelligent User Interfaces, ed. W. Gray et al., ACM, 3-10, 1993.

T. Hogg and B. A. Huberman. Better than the best: the power of cooperation.
In *SFI 1992 Lectures in Complex Systems, eds. L. Nadel and D. Stein, Addison-Wesley, 163-184, 1993.*

J. Huang, N.R. Jennings and J. Fox. Cooperation in Distributed Medical Care.
Proceedings of Second International Conference on Cooperative Information Systems, Toronto, Canada, 255-263, 1994.

J. Huang, N.R. Jennings and J. Fox. An Agent Architecture for Distributed Medical Care.
In *Intelligent Agents, eds. M.J. Wooldridge and N.R. Jennings, Lecture Notes in Artificial Intelligence, Springer-Verlag, 219-232, 1995.*

J. Huang, N.R. Jennings and J. Fox. An Agent-based Approach to Health Care Management.
Applied Artificial Intelligence: An International Journal, Taylor and Francis, London, 9:4, 401-420, 1995.

B.A. Huberman and T. Hogg - Controlling Chaos in Distributed Systems.
In *IEEE Transactions of Systems, Man, and Cybernetics*, 21:6, November/December 1991.

B.A. Huberman and T. Hogg - The Emergence of Computational Ecologies.
In *SFI 1992 Lectures in Complex Systems, eds. L. Nadel and D. Stein, Addison-Wesley, 163-184, 1993.*

N.R. Jennings. Cooperation in Industrial Systems.
Proceedings of ESPRIT Conference, Brussels, Belgium, 253-263, 1991.

N.R. Jennings, J.M. Corera and I. Laresgoiti. Developing Industrial Multi-Agent Systems.
First International Conference on Multi-Agent Systems, San Francisco, CA, 423-430, June 1995.

N.R. Jennings and A.J. Jackson. Agent-based Meeting Scheduling: A Design and Implementation.
Electronics Letters, The Institution of Electrical Engineering, 3:5, 350-352, March 1995.

R.E. Kahn and V.G. Cerf. An open architecture for a digital library system and a plan for its development.
Technical Report, Corporation for National Research Initiatives, March 1988.

H.A. Kautz and E.P.D. Pednault. Planning and plan recognition.
AT&T Technical Journal, 67:1, 25-40, Jan.-Feb.1988.

H.A. Kautz, B. Selman, M. Coen and S. Ketchpel. An Experiment in the Design of Software Agents.
In *Software Agents - Papers from the 1994 Spring Symposium, ed. O. Etzioni, AAAI Press, 43-48, 1994.*

A. Kay. User Interface: A Personal View.
In *The Art of Human-Computer Interface Design, ed. B. Laurel, Addison-Wesley, 191-208, 1994.*

A.R. Kaye and G.M. Karam. Cooperating knowledge-based assistants for the office.
ACM Transactions on Office Information Systems, 5:4, 297-326, 1987.

L. Kleinrock and A. Nilsson. On Optimal Scheduling Algorithms for Time-Shared Systems.

Journal of the ACM, 28:3, July 1981.

C.A. Knoblock and Y. Arens. An Architecture for Information Retrieval Agents.
In Software Agents - *Papers from the 1994 Spring Symposium, ed. O. Etzioni, AAAI Press, 49-56, 1994.*

R.A. Kowalski. Using meta-logic to reconcile reactive with rational agents.
To appear in "Meta-logics and Logic Programming", eds. K. Apt and F. Turini, MIT Press, 1995.

R. Kozierok and P. Maes. A learning interface agent for scheduling meetings.
In *Proceedings of the 1993 International Workshop on Intelligent User Interfaces, eds. W.D. Gray et al., ACM, 81-8, 1992.*

K. Kuwabata, T. Ishida and N. Osato. AgenTalk: Coordination Protocol Description for Multiagent Systems.
In *Proceedings of ICMAS-95, ed. V. Lesser, AAAI Press, 455, 1995.*

K.-Y. Lai and T.W. Malone. Object lens: letting end-users create cooperative work applications.
In *Human Factors in Computing Systems: Reaching Through Technology - CHI'91, eds. S.P. Robertson et al., ACM, New York, 425-6, 1991.*

A. Lawrence. Agents of the Net.
New Scientist, 1986, 34-37, 15 July 1995.

A.Y. Levy, Y, Sagiv and D. Srivastava. Toward Efficient Information Gathering Agents.
In *Software Agents - Papers from the 1994 Spring Symposium, ed. O. Etzioni, AAAI Press, 64-70, 1994.*

H. Lieberman. Mondrian: A teachable graphical editor.
In *Watch what I do: Programming by Demonstration, ed. A. Cypher, MIT Press, Cambridge, Mass., 1993.*

U. Mukhopadhyay, L.M. Stephens, M.N. Huhns and R.D. Bonnell. An Intelligent System for Document Retrieval in Distributed Office Environments.
In *Readings in Distributed Artificial Intelligence, ed. A. Bond and L. Gasser, Morgan Kaufmann, San Mateo, 565-577, 1988.*

P. Maes, editor. *Designing Autonomous Agents.*
MIT Press, Cambridge, MA, 1990.

P. Maes. Situated Agents Can Have Goals.
In *Robotics and Autonomous Systems,* 6, North Holland, 49-70, 1990.

P. Maes. Behaviour-based artificial intelligence.
In *From animals to animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behaviour, ed. J.-A. Meyer et. al., 1992.*

P. Maes. Learning Behaviour Networks from Experience.
In *Toward a Practice of Autonomous Systems - Proceedings of the First European Conference on Artificial Life, ed. F.J. Varela and P. Bourgine, MIT Press, 1992.*

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

P. Maes and R. Kozierok. Learning Interface Agents.
In *Proceedings of AAAI-93, AAAI Press/The MIT Press, 459-464, 1993.*

R. Maclin and J.W. Shavlik. Incorporating Advice into Agents that Learn from Reinforcements.
In *Proceedings of the Twelfth National Conference on Artificial Intelligence, 1994.*

J. Malec. A Unified Approach to Intelligent Agency.
In *Intelligent Agents, eds. M.J. Wooldridge and N.R. Jennings, Lecture Notes in Artificial Intelligence, Springer-Verlag, 233-245, 1995.*

T.W. Malone, K.R. Grant, F.A. Turbak, S.A. Brobst and M.D. Cohen. Intelligent information-sharing systems.
Communications of the ACM, 30:5, 390-402, May 1987.

T.W. Malone, R.E. Fikes, K.R. Grant and M.T. Howard. Enterprise: A Market-like Task Scheduler for Distributed Computing Environments.
In *The Ecology of Computation, ed. B. Huberman, North-Holland, 1988.*

T.W. Malone, K.-Y. Lai and C. Fry. Experiments with Oval: a radically tailorable tool for cooperative work.
In *Proceedings of ACM CSCW'92 Conference on Computer Supported Cooperative Work, 289-297, 1992.*

F.G. McCabe and K.L. Clark. April - Agent PRocess Interaction Language.
In *Intelligent Agents: Proceedings of ECAI-94, ed. M. Wooldridge and N. Jennings, Springer-Verlag, 324-340, 1995.*

J. McCarthy. Programs with common sense (1959).
In *Formalizing Common Sense, ed. Vladimir Lifschitz, Ablex, Norwood, NJ, 1990.*

S.L. McGregor. Prescient agents.
In D. Coleman, editor, *Proceedings of Groupware-92, 228-230, 1992.*

M. McElligott and H. Sorensen. An Evolutionary Connectionist Approach to Personal Information Filtering.
In *Proceedings of Irish Neural Networks Conference '94, University College Dublin, September 1994.*

C. Nass, J. Steuer and E.R. Tauber. Computers are Social Actors.
Human Factors in Computing Systems: CHI'94, 72-78, 1994.

C. Nass, Y. Moon, B.J. Fogg, B. Reeves and C. Dryer. Can Computer Personalities be Human Personalities?
Human Factors in Computing Systems: CHI'95, 228-229, 1995.

R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator and W. Swartout. Enabling technology for knowledge sharing.
In *AI Magazine, Fall, 1991.*

S. Nirenberg and V. Lesser. Providing Intelligent Assistance in Distributed Office Environments.
In *Readings in Distributed Artificial Intelligence, ed. A. Bond and L. Gasser, Morgan Kaufmann, San Mateo, 590-598, 1988.*

H. Nonogaki and H. Ueda. Friend21 project: a construction of 21st century human interface.
In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems, 407-414, 1991.*

M. Palaniappan, N. Yankelovich, G. Fitzmaurice, A. Loomis, B. Haan, J. Coombs and N. Meyrowitz. The Envoy framework: an open architecture for agents.
ACM Transactions on Information Systems, 10:3, 233-264, July 1992.

T.R. Payne, P. Edwards and C.L. Green. Experience with Rule Induction and k-Nearest Neighbour Methods for Interface Agents that Learn.
In *Proceedings of ML'95 Workshop on Agents that Learn from other Agents, Tahoe City, California, July 1995.*

A.S. Rao and M.P. Georgeff. BDI-agents: from theory to practice.
In *Proceedings of the First International Conference on Multiagent Systems, San Francisco, 1995.*

A. Reinhardt. Smart Networks.
BYTE, 19:10, 51-64, October 1994.

S.J. Rosenschein and L.P. Kaelbling. The synthesis of digital machines with provable epistemic properties.

In *Proceedings of Conference on Theoretical Aspects of Reasoning about Knowledge, Monterey, CA, 83-86, 1986.*

D. Rus and D. Subramanian. Designing structure-based information agents.
In *Software Agents - Papers from the 1994 Spring Symposium, ed. O. Etzioni, AAAI Press, 79-86, 1994.*

J.C. Schlimmer and L.A. Hermens. Software Agents: Completing Patterns and Constructing User Interfaces.
Journal of Artificial Intelligence Research, 1, 61-89, 1993.

J.C. Schlimmer and L.A. Hermens. A software agent for note taking.
In *Software Agents - Papers from the 1994 Spring Symposium, ed. O. Etzioni, AAAI Press, 118-121, 1994.*

T. Selker. Coach: A Teaching Agent that Learns.
Communications of the ACM, 37:7, 92-99, July 1994.

N. Shardlow. Action and agency in cognitive science.
Master's thesis, Department of Psychology, Manchester, 1990.

Y. Shoham. Agent-Oriented Programming.
Artificial Intelligence, 60:1, 51-92, 1993.

R.G. Smith. The contract net protocol: high-level communication and control in a distributed problem solver.
IEEE Trans. Comput., 29, 1104-1113, 1980

G.R. Stearns. Agents and the HP NewWave Application Program Interface.
Hewlett-Packard Journal, 40, 32-37, August 1989.

P. Stenton. Designing cooperative interfaces: Tailoring the channel.
AAAI Spring Symposium on Knowledge Based Human Computer Interaction, Stanford University, 1-4, March 1990.

P.D. Stotts and R. Furuta. Dynamic adaptation of hypertext structure.
In *Proceedings of ACM Hypertext '91, 219-231, 1991.*

J.W. Sullivan and S.W. Tyler (editors). Intelligent User Interfaces.
ACM Press, 1991

M. Tambe. Recursive agent and agent-group tracking in a real-time dynamic environment.
In *Proceedings of the First International Conference on Multiagent Systems (ICMAS'95), San Francisco, CA, June 1995.*

S.R. Thomas. The PLACA agent programming language.
In *Intelligent Agents - Proceedings of the 1994 Workshop on Agent Theories, Architectures, and Languages, ed. M. Wooldridge and N.R. Jennings, Springer-Verlag, 355-371, 1995.*

C. Toomey and R. Johnson. Software Agents for Automating Multiple-Tool Tasks.
In *Software Agents - Papers from the 1994 Spring Symposium, ed. O. Etzioni, AAAI Press, 122-125, 1994.*

E.M. Vorhees. Software Agents for Information Retrieval.
In *Software Agents - Papers from the 1994 Spring Symposium, ed. O. Etzioni, AAAI Press, 126-129, 1994.*

W. Wahlster. Intelligent interfaces as cooperative agents: from stick shift to automatic transmission in human-computer interaction.
In *Information Processing '89 - Proceedings of the IFIP 11th World Computer Congress, ed. G.X. Ritter, North-Holland, Amsterdam, 1989.*

W. Wahlster, E. Andre, W. Finkler, H.-J. Profitlich, and T. Rist. Plan-based integration of natural language and graphics generation.
Artificial Intelligence, 63:1-2, 387-427, October 1993.

C.A. Waldspurger *et al. - Spawn: A Distributed Computational Economy.*
In *IEEE Transactions on Software Engineering,* 18:2, February 1992.

M.A. Walker. Natural Language in a Desktop Environment.
Proceedings of Human Computer Interaction International Conference, 1989.

D.A. Waterman. A rule-based approach to knowledge acquisition for man-machine interface programs.
International Journal of Man-Machine Studies, 10:6, 693-711, 1978.

J.E. White. Telescript Technology: Scenes from the Electronic Marketplace.
General Magic White Paper, 1994.

J.E. White. Telescript Technology: The Foundation for the Electronic Marketplace.
General Magic White Paper, 1994.

S. Whittaker. Next generation interfaces.
AAAI Spring Symposium on Knowledge Based Human Computer Interaction, Stanford University, 127-131, March 1990.

R. Wilensky, D.N. Chin, M. Luria, J. Martin, J. Mayfield and D. Wu. The Berkeley Unix Consultant project.
Computational Linguistics, 14:4, 35-84, December 1988.

M.J. Wooldridge and N.R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review,* 10:2, 115-152, 1995a.

M.J. Wooldridge and N.R. Jennings, editors. *Intelligent Agents - Theories, Architectures, and Languages.*
Lecture Notes in Artificial Intelligence, Springer-Verlag, 890, 1995b.

G. Zlotkin and J. Rosenschein. Coalition, cryptography, and stability: mechanisms for coalition formation in task oriented domains.
In *Software Agents - Papers from the 1994 Spring Symposium, ed. O. Etzioni, AAAI Press, 87-94, 1994.*