Report

A Brief Introduction to Software Agent Technology[1]

Authors:

Hyacinth Nwana & Divine Ndumu

Approved By:

Nader Azarmi

Authorised By:

Nader Azarmi (Project Manager)

MIDAS

Intelligent Systems Research
Advanced Applications and Technology

[1] Paper presented at: *Unicom Seminar on* "Real-World Applications of Intelligent Agent Technology", pp. 278-292, 11-13 June 1996, London UK.

Abstract

Software agent technology is a rapidly developing area of research. However, there is a truly heterogeneous body of work being carried out under the 'agents' banner. This paper introduces software agent technology by overviewing the various agent types currently under investigation by researchers.

1 Introduction

Software agent technology is a rapidly developing area of research. Application domains in which agent solutions are being applied or researched into include workflow management, telecommunications network management, air-traffic control, business process re-engineering, data mining, information retrieval/management, electronic commerce, education, personal digital assistants (PDAs), e-mail filtering, digital libraries, command and control, smart databases, and scheduling/diary management. Indeed, as Guilfoyle (1995) notes

> "in 10 years time most new IT development will be affected, and many consumer products will contain embedded agent-based systems".

In this paper, we overview the evolving area of software agents. The overuse of the word 'agent' has tended to mask the fact that, in reality, there is a truly heterogeneous body of research being carried out under this banner. This paper overviews the rationales, hypotheses, goals and state-of-the-art demonstrators of the various agent types currently under investigation by researchers.

2 What is an agent?

We have as much chance of agreeing on a consensus definition for the word 'agent' as artificial intelligence (AI) researchers have of arriving at one for 'artificial intelligence'! Arguably, software agents date back to the early days of AI work, to Carl Hewitt's (1977) concurrent actor model. In this model, Hewitt proposed the concept of a self-contained, interactive and concurrently-executing object which he termed an 'actor'. This object had some encapsulated internal state and could respond to messages from other similar objects.

When we really have to, we define an agent as referring to a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user. Given a choice, we would rather say it is an umbrella term which covers a range of other more specific agent types, and then go on to list and define these other agent types.

2.1 A Typology of Agents

There are several dimensions to classify *existing* software agents. Firstly, agents may be classified by their mobility, i.e. by their ability to move around some network. This yields the classes of *static* or *mobile* agents.

Secondly, they may be classed as either *deliberative* or *reactive*. Deliberative agents derive from the deliberative thinking paradigm which holds that agents possess an internal symbolic reasoning model, and they engage in planning and negotiation with other agents in order to achieve their goals. Work on reactive agents originate from research carried out by Brooks (1986) and Agre & Chapman (1987). These agents do not have any internal, symbolic models

of their environment, and they act using a stimulus/response type of behaviour by responding to the present state of the environment in which they are embedded (Ferber 1994).

Thirdly, agents may be classified along several attributes which ideally they *should* exhibit. At BT Labs, we have identified a minimal list of three: autonomy, learning and cooperation. *Autonomy* refers to the principle that agents can operate on their own without the need for human guidance, even though this would sometimes be invaluable. A key element of autonomy is proactiveness, i.e. the ability to 'take the initiative' (Wooldridge & Jennings 1995). *Cooperation* with other agents is paramount: it is the *raison d'être* for having multiple agents. Further, the communication required to ensure cooperation generally involves high-level messages. Lastly, for agents to be truly 'intelligent', they would have to *learn* as they react and/or interact with their external environment; such that with time, their performance increases. We use these three characteristics in Fig. 1 to derive three types of agents: *collaborative agents, interface agents* and truly *smart agents*.

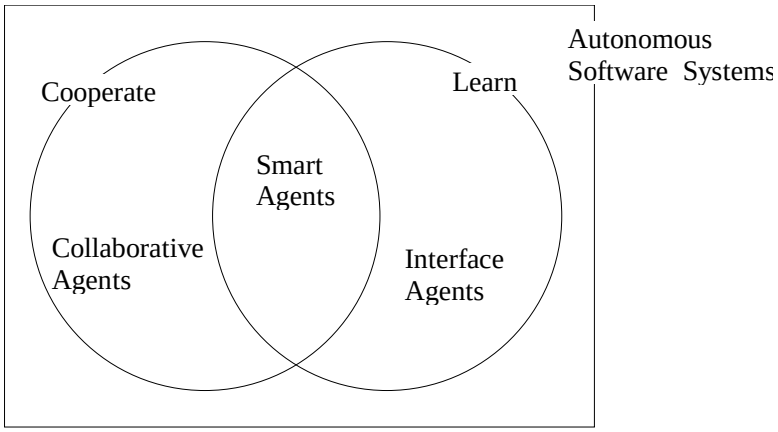

Figure 1: A part view of an agent typology

We must emphasise that these distinctions are *not* definitive. For example, with collaborative agents, there is more emphasis on cooperation and autonomy than on learning; but we do not imply that collaborative agents never learn. We do *not* consider anything else which lie outside the 'circles' to be agents. For example, most expert systems are largely autonomous; but, typically, they do not cooperate or learn. Ideally, agents should do all three equally well, but this is the *aspiration* rather than the reality. Truly smart agents do not yet exist!

Fourthly, agents may sometimes be classified by their roles (particularly, if the roles are major ones), e.g. world wide web (WWW) information gathering agents. Essentially, such agents help manage the vast amount of information in wide area networks like the internet. We refer to this class of agents as *information* or *internet agents*. Again, information agents may be static or mobile and deliberative or reactive.

Fifthly, we have also included the category of *hybrid* agents which combine two or more agent philosophies in a single agent.

There are other attributes of agents which we consider secondary to those already mentioned. For example, is an agent versatile (i.e. does it have many goals or does it engage in a variety of tasks)? Is an agent benevolent or non-helpful, antagonistic or altruistic? Does an agent lie knowingly or is it always truthful? Is it temporally continuous? Some researchers are also attributing emotional attitudes to agents — do they get 'fed up' being asked to do the same thing time and time again (Bates 1994)? Some agents are also imbued with *mentalistic*

attitudes such as beliefs, desires and intentions (Rao & Georgeff 1995; Jennings 1993). Such attributes as these provide for a stronger definition of agenthood.

In essence, *agents exist in a truly multi-dimensional space*. However, for the sake of clarity of understanding, we have 'collapsed' this space into a single list. Hence, we identify seven types of agents:

- Collaborative agents
- Interface agents
- Mobile agents
- Information/Internet agents
- Reactive agents
- Hybrid agents
- Smart Agents

There are some applications which combine agents from two or more of these categories, and we refer to these as *heterogeneous agent systems*.

3 A Panoramic Overview of the Different Agent Types

In this section we will, in turn, overview all the types of agents identified in our typology of the previous section in terms of some or all of the following: their essential metaphors, hypotheses/goals, motivations, roles, prototypical examples, and potential benefits. We do not overview smart agents on the grounds that this is the *aspiration* of agent researchers rather than the *reality*.

3.1 Collaborative Agents

As shown in Fig. 1, collaborative agents emphasise autonomy and cooperation with other agents in order to perform tasks for their owners in open and time-constrained multi-agent environments. They may learn, but this aspect is not typically a major emphasis of their operation, though some perform limited parametric or rote learning. To coordinate their activities, they may have to *negotiate* in order to reach mutually acceptable agreements. Collaborative agents tend to be static, large, coarse-grained agents. They may be benevolent, rational, truthful, some combination of these or neither. As noted earlier, some researchers are providing stronger definitions to such agents, hence, the class of collaborative agents may itself be perceived as a broad grouping.

3.1.1 Motivation and Benefits

The rationale for having collaborative agent systems is a specification of the goal of distributed AI (DAI), which may be summarised as "*creating a system that interconnects separately developed collaborative agents, thus enabling the ensemble to function beyond the capabilities of any of its members*" (Huhns & Singh 1994). Thus, DAI research, and hence collaborative agents research, aims to

- solve problems that are too large for a centralised single agent to do due to resource limitations or the sheer risk of having one centralised system.

- allow for the interconnecting and interoperation of existing legacy systems, e.g. expert systems, decision support systems, conventional programs, etc.
- provide solutions to inherently distributed problems. For instance, solutions which draw from distributed information sources, e.g. distributed on-line information sources or distributed sensor networks (cf. DVMT, Durfee *et al.* 1987); and solutions where the expertise is distributed, e.g. in health care provisioning or air-traffic control (cf. OASIS, Rao & Georgeff 1995).

The potential benefits of collaborative agents research, inherited from DAI, include modularity (which reduces complexity), speed (due to parallelism), reliability (due to redundancy) and flexibility (i.e. new tasks are composed more easily from the more modular organisation). Collaborative agents also inherit benefits from AI such as operation at the knowledge level, easier maintenance, reusability and platform independence.

3.1.2 A Prototypical Example: The Pleiades System

The Pleiades project (see URL1) applies collaborative agents in the domain of Organisational Decision Making over the "InfoSphere" (which refers essentially to a collection of internet-based heterogeneous resources).

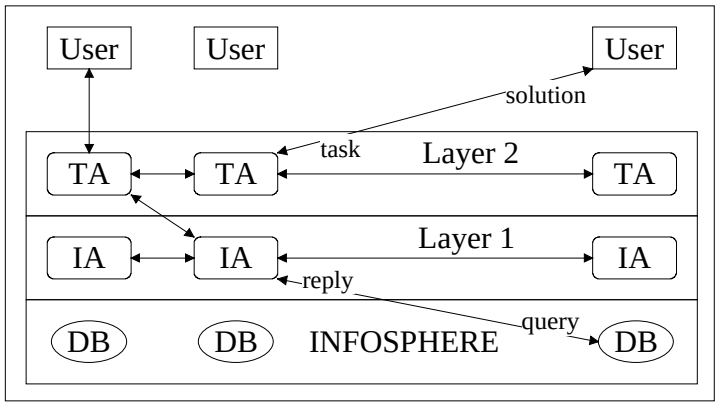

Figure 2: The Pleiades distributed system architecture (adapted from Sycara 1995)

Pleiades is a distributed collaborative agent-based architecture which has two layers of abstraction: the first layer contains *task-specific* collaborative agents and the second *information-specific* collaborative agents (Fig. 2). This architecture has been used to develop a visitor hosting system in which agents cooperate in order to create and manage a visitor's schedule to Carnegie Mellon University. In the architecture, task-specific agents perform a particular task for their users, e.g. arranging appointments and meetings with other task agents. These agents coordinate and schedule plans based on the context. They collaborate with one another (within Layer 1) in order to resolve conflicts or integrate information. In order to garner the information required at this level, they request information from information-specific agents. Information agents, in turn, may collaborate with one another (i.e. within Layer 2) in order to provide the information requested back to the Layer 1 requesting agent. The sources of the information are the many databases in the infosphere. Ultimately, the task agents propose a solution to their users.

Task agents encode a model of the task domain and knowledge of how to perform tasks, as well as an acquaintance model detailing the capabilities of other task or information agents. They also possess some learning mechanisms. Information agents possess knowledge of the

various information sources and how to access them, and an acquaintance model specifying the abilities of other information agents (Sycara 1995).

Individually, an agent consists of a planning module linked to its local beliefs and facts database. It also has a local scheduler, a coordination module and an execution monitor. Thus, agents can instanstiate task plans, coordinate these plans with other agents and schedule/monitor the execution of their local actions. Interestingly, the architecture has no central planner and hence agents must all engage in coordination by communicating to others their constraints, expectations and other relevant information.

3.1.3 A Brief Critical Review of Collaborative Agent Systems Work

There are many other useful pieces of work on collaborative agents. At BT Labs, two prototype collaborative agent-based systems, ADEPT and MII, have been developed recently. ADEPT (O'Brien & Wiegand 1996) employs collaborative agents in the application area of business process re-engineering while MII (Winter *et al*. 1996) demonstrates that collaborative agents can be used to perform decentralised management and control of consumer electronics, typically PDAs or PCs integrated with services provided by the network operator.

The key criticism of collaborative agents levelled by some researchers stems from their grounding in the deliberative thinking paradigm. Researchers in the reactive agents camp argue that this results in brittle and inflexible demonstrators with slow response times. In Section 3.5 we discuss briefly the deliberative versus reactive agents debate.

In conclusion, we see a potential major role for collaborative agents in industrial applications, as has been evidenced by, for example, the ARCHON project (Wittig 1992) which used collaborative agents in electricity transportation management, and OASIS (Rao & Georgeff 1995) where collaboraive agents are used in air-traffic control.

3.2 Interface Agents

Interface agents (cf. Fig. 1) emphasise autonomy and learning in order to perform tasks for their owners. Maes (1994) points out that the key metaphor underlying interface agents is that of a *personal assistant* who is *collaborating with the user* in the same work environment. Note the subtle emphasis and distinction between collaborating with *the user* and collaborating with *other agents* as is the case with collaborative agents.

Essentially, interface agents support and provide proactive assistance, typically to a user learning to use a particular application such as a spreadsheet or an operating system. The user's agent observes and monitors the actions taken by the user in the interface (i.e. 'watches over the shoulder of its user'), learns new 'short-cuts', and suggests better ways of doing the task. As for learning, typically, interface agents learn to assist their users in the following four ways: (i) by observing and imitating the user, (ii) through receiving positive and negative feedback from the user, (iii) by receiving explicit instructions from the user, and (iv) by asking other agents for advice. Generally, the learning modes are by rote memory-based learning or parametric, though other techniques such as evolutionary learning are also being introduced. Their cooperation with other agents, if any, is limited typically to asking for advice.

Maes (1995) specifies two preconditions to be fulfilled by suitable application programs for interface agents: firstly, that there is substantial repetitive behaviour in using the application

program (otherwise, the agent will not be able to learn anything) and, secondly, that this repetitive behaviour is potentially different for different users (otherwise, use a knowledge-based approach).

3.2.1 Benefits/Roles

The general benefits of interface agents are threefold. First, they make less work for the end user and application developer. Secondly, the agent can adapt, over time, to its user's preferences and habits. Finally, know-how among the different users in a community may be shared (e.g. when agents learn from their peers). Perhaps these will be better understood by discussing a few of the roles for which interface agents have been built.

Kozierok & Maes (1993) describe an interface agent, Calendar Agent, that assists its user in scheduling meetings. It can learn, over time, the preferences and commitments of its user, e.g. she does not like to attend meetings on Friday afternoons, she prefers meetings in the morning, etc. The learning techniques employed are memory-based learning and reinforcement learning.

Liebermann (1995) describes Letizia, a keyword and heuristic-based search agent, which assists in web browsing. Since most browsers encourage depth-first browsing, Letizia conducts a breadth-first search concurrently for other useful locations that the user may be interested in. It does this by 'guessing' the user's intention from her browsing behaviour, e.g. she keeps returning to some particular page, and proceeding to search using the search engine. By doing this, it is able to recommend some other useful serendipitous locations.

Maes (1994) describes a news filtering agent, NewT, that helps users filter and select articles from a continuous stream of Usenet Netnews. NewT agents are trained by presenting to them positive and negative examples of what should or should not be retrieved. They are message-content and keyword-based but also exploit other information such as the author and source.

There is no denying the fact that interface agents can be deployed in real applications in the short term because they are simple, operate in limited domains and require no cooperation.

3.3 Mobile Agents

3.3.1 Hypothesis, Motivation and Benefits

Mobile agents are software processes capable of roaming wide area networks (WANs) such as the WWW, interacting with foreign hosts, performing tasks on behalf of their owners and returning 'home' having performed the duties set them. These duties may range from making a flight reservation to managing a telecommunications network. Mobile agents are agents because they are autonomous and they cooperate, albeit differently to collaborative agents. For example, they may cooperate by one agent making the location of some of its internal objects and methods known to other agents.

The key hypothesis underlying mobile agents is the idea that in certain applications, they provide a number of *practical*, though *non-functional*, advantages which escape their static counterparts. For example, imagine having to download many images just to pick out one. Is it not more natural to get your agent to "go" to that location, do a local search and only transfer the chosen compressed image back across the network?[2]

[2] This example was provided by our colleague Barry Crabtree.

3.3.2 How Mobile Agents Work: A Brief Telescript View

Telescript is an interpreted object-oriented and remote programming language which allows for the development of distributed applications (URL2). Fig. 3 summarises a part view of the Telescript architecture. The Telescript Development Environment (TDE) comprises, among other things, the engine (interpreter and runtime development environment), browser, debugger and associated libraries.

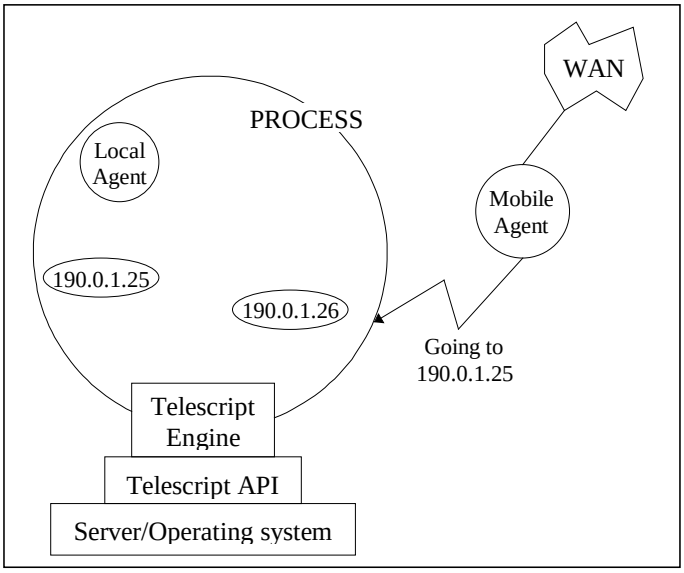

Figure 3: A part view of Telescript architecture (adapted from Wayner 1995)

Telescript applications consist of Telescript agents operating within a 'world' or cyberspace of places and engines; both of which are objects. The top class in Telescript's object hierarchy is the *process*. A Telescript engine is itself a preemptive multitasking interpreter which can run multiple processes. Hence, the engine can host multiple agents that share data/information between themselves. Furthermore, a *place* is a process which can contain an arbitrary number and depth of other places. *Agents*, unlike *places*, are objects which cannot contain other processes, but they can 'go' from place to place. An agent requiring a service defined at some given place must go to that place and call the operations there (cf. Fig. 3). Thus, "go" is the primitive which allows for inter-process communication. Two or more agent processes can *meet* in a place and make use of each other's services.

A "go" requires a destination *place* and the host engine packages up the agent along with all its data, stack and instruction pointer and ships it off to this destination place which may be across a vast WAN. At its destination, the other Telescript-enabled engine unpacks it, checks its authentication, and it is then free to resume execution at this new place. When it finishes, it returns to its original host having performed the task required by its owner.

It is also important to point out that mobile agent systems need not only be constructed using a remote programming language like Telescript. Indeed, Appleby & Steward (1994) at BT Labs prototyped an award-winning C/C++ programmed mobile agent-based system for controlling telecommunications networks. Other languages to consider include Java, Xlisp, Agent-Tcl and Safe-Tcl.

3.3.3 Mobile Agent Applications

Mobile agent applications do not currently abound but are likely to increase in the future. However, the first commercial application was Sony's Magic Link PDA or personal intelligent communicator (see URL3). Essentially, it assists in managing a user's e-mail, fax, phone and pager as well as linking the user to Telescript-enabled messaging and communication services such as America Online and AT&T PersonaLink Services.

Plu (1995) mentions that France Telecom has prototyped some services based on Telescript. In one of their demonstrators, mobile Telescript agents integrate railway ticketing and car renting services. IBM plans to launch their Communication System (Reinhardt 1994) which uses mobile agents for providing a communications super-service capable of routing and translating communications from one service and medium to another, e.g. mobile to desktop, PDA to fax, speech to text, etc.

3.4 Information/Internet Agents

3.4.1 Hypothesis, Motivation and Benefits

Information agents perform the role of managing, manipulating or collating information from many distributed sources. The motivation for developing information agents is at least twofold. Firstly, there is simply a need for tools to manage the information explosion of the WWW. Everyone on the WWW would benefit from them in the same way as they are benefiting from search facilitators such as Spiders, Lycos or Webcrawlers. Secondly, there are vast financial benefits to be gained. Recall that Netscape Corporation grew from relative obscurity to a billion dollar company almost overnight — and a Netscape or Mosaic client simply offers general browsing capabilities, albeit with a few add-ons. Whoever builds the first useable Netscape equivalent of a proactive, dynamic, adaptive and cooperative WWW information manager is certain to reap enormous financial rewards.

Before we proceed, we should clarify that there is a rather fine distinction, if any, between information agents and some of those which we have earlier classed as interface or collaborative agents (e.g. NewT, Letizia, etc.). In this text, information agents are defined by what *they do*, in contrast to collaborative or interface agents which we defined by what *they are*.

3.4.2 How Information Agents Work

Internet agents could be mobile; however, this is not the norm as yet. Typical static ones are embedded within an internet browser and use a host of internet management tools such as Spiders and search engines in order to gather the information. Etzioni & Weld (1994) describe a state-of-the-art internet agent called the internet softbot (software robot). It allows a user to make a high-level menu-based request such as "send the budget memos to Mitchell at CMU" and "Get all of Ginsberg's technical reports that aren't stored locally", and the softbot is able to use search and inference knowledge to determine *how* to satisfy the request in the internet. In doing so, it is able to tolerate ambiguity, omissions and the inevitable errors in the user's request. Etzioni & Weld use a strong analogy to a real robot in order to describe their softbot's interface to the internet. For example, they describe the softbot's *effectors* to include `ftp`, `telnet`, `mail` and numerous file manipulation commands such as `mv` or `compress`. The *sensors* provide the softbot with information about the external world and they include internet facilities such as `archie`, `gopher` and `netfind`.

3.4.3 A Brief Critical Review of Information Agents Work

We expect information agents to be a major growth area in the next couple of years. At BT Labs, Davies & Weeks (1995) designed and implemented the Jasper agent. Jasper agents work on behalf of a user or a community of users, and are able to store, retrieve, summarise and inform other agents of information useful to them found on the WWW. As a user works with her Jasper agent, a profile of her interests is built dynamically based on keywords. A Jasper agent is able to suggest interesting WWW pages to a user by matching her profile with those of other users in the community. A successful match results in the user being told of other WWW pages that her peers find 'interesting'.

The key problem with static information agents is in keeping their indexes up-to-date in an environment which is prone to complete chaos. Thus, it is probable that the majority of future information agents will be of the mobile variety: they would be able to navigate the WWW and store its topology in a database, say, at their home site. The local database may then be locally queried by the user.

3.5 Reactive Software Agents: An Overview

Also often referred to as autonomous agents, reactive agents represent a special category of agents which do *not* possess internal, symbolic models of their environments; instead they respond in a stimulus-response manner to the present state of the environment in which they are embedded. Reactive agents work dates to research such as Brooks (1986) and Agre & Chapman (1987), but many theories, architectures and languages for these sorts of agents have been developed since.

Maes (1991a) highlights the three key ideas which underpin reactive agents. Firstly, 'emergent functionality': reactive agents are relatively simple and they interact with other agents in basic ways. Nevertheless, complex patterns of behaviour *emerge* from these interactions when the ensemble of agents is viewed globally. Hence, there is no *a priori* specification (or plan) of the behaviour of the set-up of reactive agents. Secondly, 'task decomposition': a reactive agent is viewed as a collection of modules which operate autonomously and are responsible for specific tasks (e.g. sensing, motor control, computations, etc.). Communication between the modules is minimised and of quite a low-level nature. No global model exists within any of the agents and, hence, the global behaviour has to emerge. Thirdly, reactive agents *tend* to operate on representations which are close to raw sensor data, in contrast to the high-level symbolic representations that abound in the other types of agents discussed so far.

3.5.1 Hypothesis, Motivation and Benefits

The essential hypothesis of reactive agent-based systems is a specification of the *physical grounding hypothesis*. The *physical symbol system hypothesis*, which underpins classical AI, holds that for a physical system to demonstrate intelligent action it should be a symbol processing system. The physical grounding hypothesis challenges this long-held view arguing it is flawed fundamentally, and that it imposes severe limitations on symbolic AI-based systems. This new hypothesis states that in order to build a system that is intelligent, it is necessary to have representations grounded in the physical world (Brooks 1991a). Brooks argues that this hypothesis obviates the need for symbolic representations or models because the world becomes its own best model. Furthermore, this model is always kept up-to-date since the system is connected to the world via sensors and/or actuators. Hence, the reactive

agents hypothesis may be stated as follows: intelligent agent systems can be developed from simple agents which do not have internal symbolic models, and whose 'intelligence' derives from the emergent behaviour of the interactions of the various agents.

The key benefits which motivates reactive agents work is the hope that they would be more robust and fault tolerant than other agent-based systems, e.g. a single agent in an ensemble may be lost without any catastrophic effects. Other benefits include flexibility and adaptability in contrast to the inflexibility, slow response times and brittleness of classical AI systems.

3.5.2 Reactive Agent Applications

There are relatively few reactive software agent-based applications. Partly, due to this reason, there is no standard mode to their operation; rather, they tend to depend on the reactive agent architecture chosen.

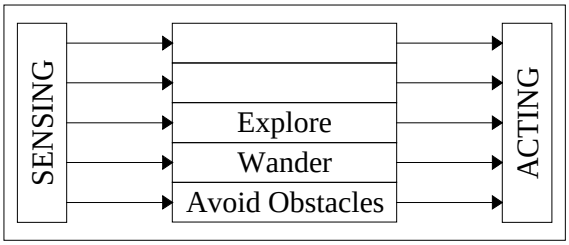

Figure 4: Brooks' subsumption architecture

Perhaps the most celebrated reactive agent architecture is Brooks' (1991b) *subsumption architecture*. The architecture consists of a set of modules, each of which is described in a subsumption language based on augmented finite state machines (AFSM). An AFSM is triggered into action if its input signal exceeds some threshold, though this is also dependent on the values of suppression and inhibition signals into the AFSM. Note that AFSMs represent the only processing units in the architecture, i.e. there are no symbols as in classical AI work. The modules are grouped and placed in layers (which work asynchronously) such that modules in a higher level can inhibit those in lower layers (see Fig. 4). Each layer has a hard-wired purpose or *behaviour*, e.g. to avoid obstacles or to enable/control wandering. This architecture has been used to construct, at least, ten mobile robots.

Arguably, the most basic reactive architecture is that based on situated-action rules. A situated action agent acts essentially in ways which is 'appropriate' to its situation, where 'situation' refers to a potentially complex combination of internal and external events and states (Connah 1994). Situated-action 'agents' have been used in PENGI, a video game (Agre & Chapman 1987), and by Ferber (1994) to simulate ant societies where each ant is modelled as an agent, and a limited ecosystem composed of biotapes, shoals of fish and fishermen.

3.6 Hybrid Agents

3.6.1 Hypothesis, Motivation and Benefits

So far, we have reviewed five types of agents: collaborative, interface, mobile, internet and reactive agents. The debates as to which of them is 'better' are academic, sterile and rather too early to get into. Since each type has (or promises) its own strengths and deficiencies, the trick, as always, is to maximise the strengths and minimise the deficiencies of the most

relevant technique for your particular purpose. Frequently, one way of doing this is to adopt a *hybrid* approach, like Maes (1991b), which brought together some of the strengths of both the deliberative and reactive paradigms. In such a case the reactive component, which would take precedence over the deliberative one, brings about the following benefits: robustness, faster response times and adaptability. The deliberative part of the agent would handle the longer term goal-oriented issues. Hence, hybrid agents refer to those whose constitution is a combination of two or more agent *philosophies* within a singular agent.

3.6.2 Hybrid Agent Architectures

As is the case with reactive agents, there are few hybrid agent architectures. Typically, however, they have a layered architecture as is evidenced by Muller *et al*.'s (1995) InteRRaP architecture.

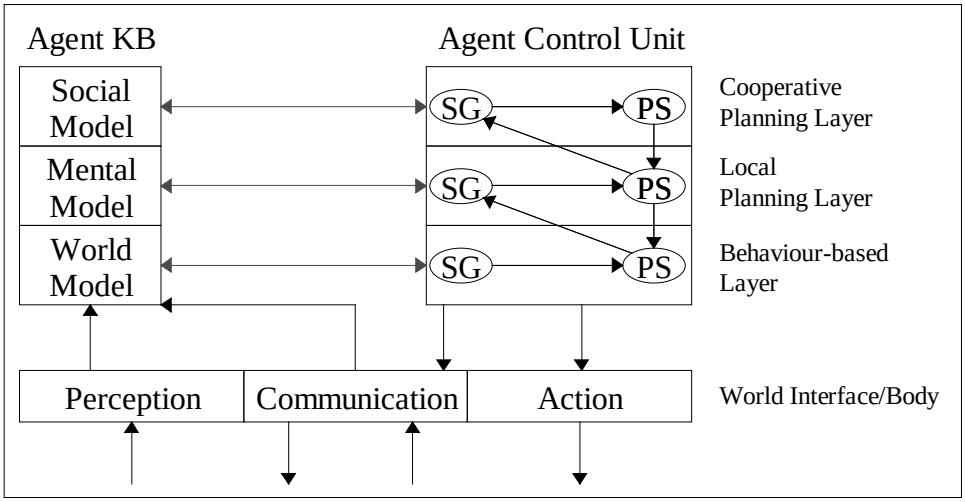

Figure 5: The InteRRaP hybrid architecture (adapted from Fischer *et al*. 1996)

Muller *et al*.'s InteRRaP architecture (Fig. 5) comprises three control layers: the behaviour-based layer (BBL), the local planning layer (LPL) and the cooperative planning layer (CPL). The reactive part of the framework is implemented by the BBL which contains a set situation-action rules. These describe the agent's reactive skills which implements fast situation recognition in order to react to time-critical situations. The intermediate LPL implements local goal-directed behaviour while the topmost CPL enables the agent to plan/cooperate with other agents in order to achieve multi-agent plans, as well as resolve conflicts. LPL and CPL allow for more deliberation. These layers all work asynchronously with different models in the agent's knowledge base: BBL, LPL and CPL operate with the world, mental and social models respectively. The InteRRaP architecture has been evaluated by constructing a FORKS application which simulates forklift robots working in an automated loading dock environment.

3.7 Heterogeneous Agent Systems

3.7.1 Hypothesis, Motivation and Benefits

Heterogeneous agent systems refer to an integrated set-up of at least two or more agents which belong to two or more different agent classes. A heterogeneous agent system may also contain hybrid agents. Genesereth & Ketchpel (1994) articulate clearly the motivation for heterogeneous agent systems. The essential argument is that the world abounds with a rich

diversity of software products. Though these programs work in isolation, there is an increasing demand to have them *interoperate* — hopefully, in such a manner that they provide 'added-value' as an ensemble than they do individually. Indeed, a new domain called *agent-based software engineering* has been invented in order to facilitate the interoperation of miscellaneous software agents. A key requirement for interoperation amongst heterogeneous agents is having an agent communication language (ACL), with an agent-independent semantics, through which the different software 'agents' can communicate with each other.

3.7.2 How Heterogeneous Agent Systems Work

To commence, we provide the rather *specific* definition of the word 'agent' proffered in agent-based software engineering. It defines a software agent as such "if an only if it communicates correctly in an agent communication language" (Genesereth & Ketchpel 1994). If new agents are constructed such that they abide by this dictum, then putting them together in a heterogeneous set-up is possible, though not trivial. However, with legacy software, they need to be converted into software agents first. The latter authors note that there are three ways of doing this conversion. Firstly, the legacy software may totally be rewritten — a most costly approach. Secondly, a *transducer* approach may be used. The transducer is a *separate* piece of software which acts as an interpreter between the agent communication language and the legacy software's native communication protocol. This is the favoured approach in many situations where the legacy code may be too delicate to tamper with or is unavailable. Lastly, is the *wrapper* technique in which some code is "injected" into the legacy program in order to allow it communicate in ACL. The wrapper can directly access and modify the program's data structures. This is clearly a more interventionist approach, but offers greater efficiency than the transduction approach.

Once the agents are available, there are two possible architectures to choose from: one in which all the agents handle their own coordination or another in which groups of agents can rely on special system programs to achieve coordination. The disadvantage of the former is that the communication overhead does not ensure scalability. As a consequence, the latter federated approach is typically preferred. Here, agents do not communicate directly with one another but do so through intermediaries called *facilitators*. Essentially, the agents surrender some of their autonomy to the facilitators who are able to locate other agents on the network capable of providing various services. The facilitators also establish the connections across the environments and ensure correct 'conversation' amongst agents; ARCHON (Wittig 1992) and PACT (Cutkosky *et al*. 1993) used such an architecture.

This concludes our panoramic overview of the different classes of agents identified in Section 2.

4 Conclusion

"Smart agents are here to stay. Once unleashed, technologies do not disappear" (Norman 1994).

This paper has pilfered from a diverse literature in order to overview the rapidly evolving area of software agents. We have overviewed the area from the viewpoint of the clear diversity of agents being researched into in universities and research laboratories worldwide. We hope this paper provides a useful contribution to understanding this exciting field of software agents. Nwana (1996) presents an extended more comprehensive version of this paper.

Acknowledgements

We would like to thank the members of the Intelligent Systems Unit, BT Laboratories, who provided feedback which has improved the quality of this paper. The authors bear all responsibility for any misunderstandings and/or errors therein. This work was funded by BT Laboratories.

References

Agre P. E. & Chapman D. (1987) "Pengi: An Implementation of a Theory of Activity", *Proc 6th National Conf on Artificial Intelligence*, San Mateo, CA: Morgan Kaufmann, 268-272.

Appleby S. & Steward S. (1994) "Mobile Software Agents for Control in Telecommunications Networks", *BT Technol J* **12** (2) 104-113.

Bates J. (1994) "The Role of Emotion in Believable Characters", *Communications of the ACM* **37** (7), 122-125.

Brooks R. A. (1986) "A Robust Layered Control System for a Mobile Robot", *IEEE J Robotics and Automation* **2** (1), 14-23.

Brooks R. A. (1991) "Elephants Don't Play Chess", In Maes P. (ed) (1991) *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, London: The MIT Press, 3-15.

Brooks R. A. (1991) "Intelligence without Representation", *Artificial Intelligence* **47**, 139-159.

Chaib-draa B., Moulin B., Mandiau R. & Millot P. (1992) "Trends in Distributed Artificial Intelligence", *Artificial Intelligence Review* **6**, 35-66.

Connah D. (1994) "The Design of Interacting Agents for Use in Interfaces", in Brouwer-Janse D. & Harringdon T. L. (eds.), *Human-Machine Communication for Educational Systems Design, NATO ASI Series,* Series **F**, Computer and Systems Sciences **129**, Heidelberg: Springer Verlag.

Cutkosky M. R., Engelmore R. S., Fikes R. E. Genesereth M. R., Gruber T. R. Tenenbaum J. M. & Weber J. C. (1993) "PACT: An Experiment in Integrating Concurrent Engineering Systems", *IEEE Computer* **1**, January, 28-37.

Davies N. J. & Weeks R. (1995) "Jasper: Communicating Information Agents", In *Proc 4th Int Conf on the World Wide Web*, Boston, USA, December.

Durfee E. H., Lesser V. R. & Corkill D. (1987) "Coherent Cooperation among Communicating Problem Solvers", *IEEE Transactions on Computers* **C-36**(11), 1275-1291.

Etzioni O. & Weld D. (1994) A Softbot-Based Interface to the Internet", *Communications of the ACM* **37** (7), 72-76.

Ferber J. (1994) “Simulating with Reactive Agents”, In Hillebrand, E. & Stender J. (Eds.) *Many Agent Simulation and Artificial Life*, Amsterdam: IOS Press, 8-28.

Fischer K., Muller J. P. & Pischel M. (1996) “Unifying Control in a Layered Agent Architecture”, *Technical Report TM-94-05*, German Research Center for AI - (DFKI GmbH).

Genesereth M. R. & Ketchpel S. P. (1994) “Software Agents”, *Communications of the ACM* **37** (7), 48-53.

Guilfoyle C. (1995) “Vendors of Agent Technology”, *UNICOM Seminar on Intelligent Agents and their Business Applications*, 8-9 November, London, 135-142.

Hewitt C. (1977) “Viewing Control Structures as Patterns of Passing Messages”, *Artificial Intelligence* **8**(3), 323-364.

Huhns M. N. & Singh M. P. (1994) “Distributed Artificial Intelligence for Information Systems”, CKBS-94 Tutorial, June 15, University of Keele, UK.

Jennings N. R. (1993) “Specification and Implementation of a Belief Desire Joint-Intention Architecture for Collaborative Problem Solving”, *J Intelligent and Cooperative Information Systems* **2**(3), 289-318.

Kozierok R. & Maes P. (1993) “A Learning Interface Agent for Scheduling Meetings”, *Proc ACM-SIGCHI Int Workshop on Intelligent User Interfaces*, Florida, 81-93.

Lieberman H. (1995) “Letizia: An Agent that Assists Web Browsing”, In *Proc IJCAI 95*, AAAI Press.

Maes P. (1991) (ed) *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, London: The MIT Press.

Maes P. (1991) “Situated Agents Can Have Goals”, In Maes P. (ed) (1991) *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, London: The MIT Press, 49-70.

Maes P. (1994) “Agents that Reduce Work and Information Overload”, *Communications of the ACM* **37** (7), 31-40.

Maes P. (1995) “Intelligent Software”, *Scientific American* **273** (3), September.

Muller J. P., Pishel M. & Thiel M. (1995) “Modelling Reactive Behaviour in Vertically Layered Agent Architectures”, In Wooldridge M. & Jennings N. (eds.) (1995) *Intelligent Agents*, Lecture Notes in Artificial Intelligence **890**, Heidelberg: Springer Verlag, 261-276.

Norman D. (1994) “How might people interact with agents” *Communications of the ACM* **37** (7), 68-76.

Nwana H. S. (1996) “Software Agents: an overview”, to appear *Knowledge Engineering Review*.

O’Brien P. & Wiegand M. (1996) “Agents of Change in Business Process Management”, *BT Technol J*, October, To appear.

Plu M. (1995) “Software Agents in Telecommunications Network Environments”, *UNICOM Seminar on Intelligent Agents and their Business Applications*, 8-9 November, London, 225-243.

Rao A. S. & Georgeff M. P. (1995) “BDI Agents: From Theory to Practice”, In *Proc 1st Int Conf on Multi-Agent Systems (ICMAS-95)*, San Francisco, USA, June, 312-319.

Reinhardt A. (1994) “The Network with Smarts”, *Byte*, October, 51-64.

Sycara K. (1995) “Intelligent Agents and the Information Revolution”, *UNICOM Seminar on Intelligent Agents and their Business Applications*, 8-9 November, London, 143-159.

URL1: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-5/www/pleiades.html.

URL2: http://www.genmagic.com.

URL3: http://www.sel.sony.com.

Wayner P. (1995) *Agents Unleashed: A Public Domain Look at Agent Technology*, Boston, MA: AP Professional.

Winter C. S., Titmuss R. & Crabtree B. (1996) "Intelligent agents, mobility and multimedia information", *Proc 1st Int Conf on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM '96)*, London, 22-24 April.

Wittig T. (1992) (ed.) *ARCHON: An Architecture for Multi-Agent Systems*, London: Ellis Horwood.

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