*The Knowledge Engineering Review*, Vol. 13:2, 1998, 151–156
*Printed in the United Kingdom*


# What Chance Software Agents?

BARRY CRABTREE

*BT Laboratories, Martlesham Heath, Ipswich IP5 7RE, UK. Email: Barry.Crabtree@bt.com*

**Abstract**

Software agents have come a long way in the last ten or so years. This paper looks at the practical challenges many software agent developers face including: interfacing to legacy systems; developing distributed systems; agent competence; and trust and privacy issues. Because it covers such a broad area, it will focus on how the problems and issues have changed over the years, and highlight some of the perennial problems.

## 1 Introduction

Software agents are beginning to emerge in products and services. They fall short of the promises from the early (and continuing) hype, not unlike the claims for AI & expert systems, but they represent helpful, real applications. The World Wide Web provides the main application infrastructure for these applications as demonstrated by companies such as Autonomy[1], OpenSesame[2], Jango[3] and AgentSoft[4]. Let us hope that the continuing hype – and inevitable backlash will not stop these systems from becoming commonplace and revolutionize the way we use computer systems.

Software agents are not a new term for software systems – although many people could be forgiven for thinking that, but are essentially software systems that can have tasks delegated to them. The agents themselves decide how and when they should perform the task, and some agents have the ability to negotiate the task or even reject the task. The diverse types of software agent are covered in depth in Nwana (1996).

The personal assistant agents championed by Pattie Maes (1994) have received the most attention over the years mainly because they strike a chord with almost any computer user. The idea is that these personal assistants would take over some of the more mundane, repetitive tasks in the office environment. The agents would ''look over your shoulder'' and monitor how you performed certain tasks, when they were confident how you would react in certain situations, they take over or assist in that task. These could be agents to look after your e-mail: filtering out the low priority or junk mail; agents that managed your diary; or agents to draw your attention to relevant or interesting news. These types of agents have one thing in common – they were all personalised for a particular user and adapt to the individual's behavior.

The other main form of software agents, although not as immediate to the individual as personal assistant agents, are agents that work collaboratively to solve problems. These agents tend to model the way that businesses and groups of individuals organise themselves to solve problems. Each agent possesses enough ''intelligence'' to schedule its work and delegate or negotiate with other agents to perform tasks on their behalf. These agents could represent different specialists in a business, each called upon to perform work at the right time as part of a business process for example. In another

[1] http://www.agentware.com
[2] http://www.opensesame.com
[3] http://www.jango.com
[4] http://www.agentsoft.com

scenario they could represent both network and service management systems that need to negotiate with each other to decide how best to manage network traffic under certain load conditions. In all cases, these agents need to be able to know about other agents in the society, and how they can be contacted to collaborate on task issues.

Common issues that have to be addressed in all agent systems include:

- *Tasks*: all agent based systems perform tasks on behalf of their ''owners''. This requires agents to understand task requests, translating them into the actions required to carry out their tasks, and interact appropriately with other software systems (databases, information management systems, legacy systems).
- *Agent systems development*: on the practical level, agent systems need to be designed, developed and debugged so development environments and methodologies are particularly important.
- *Agent competence*: as agents operate in an environment they should adapt and improve their performance over time. Agents that act on behalf of a single user need to take into account particular aspects of that users' needs in order to be effective. Agents that co-operate and negotiate with one another will need to learn which agents are truthful and helpful, so as to devise the appropriate negotiation strategy to be competent over time.
- *Trust and privacy*: many agents will be operating in an environment where they have access to personal and confidential information. Personal agents may be prioritising e-mails for you, so you need to be confident in their competence. Some agents may share information about you with other agents so you need to trust the agent to honour confidentiality. Collaborative agents should not negotiate to do tasks that they cannot achieve.

This paper discusses these areas in terms of some of the technical, practical and social issues that need to be addressed. It highlights places where advances have been made and where there are still problems to be addressed.

## 2 Tasks

Most agent-based systems perform tasks that necessitate interfacing to some external system. This might be to gather information as the basis for a task or control an external device. At the programming interface level there are standards such as the OMG with CORBA as a means of both finding appropriate ''services'', and being able to call them in a common manner. This frees the programmer from the burden of knowing the physical location of the service and any platform dependant issues. This, for example, could provide the infrastructure for the wrapper approach to legacy systems covered in Genesereth (1994), and more recently by the wrappers standards in FIPA[5]. The World Wide Web hosts many general information providing sites such as train timetables, financial services and price lists for numerous goods. Because of the many services available, there are many systems attempting to use them, and it has started up research efforts to try and induce interface wrappers semi-automatically (Ashish, 1987; Kushmerick, 1997). Depending on the system that the agents interface to, there are two options for the interface:

- Through a well defined API (c.f. CORBA).
- *Ad hoc* methods such as terminal emulation, screen-scraping and content analysis.

Unfortunately, many legacy systems do not have a well-defined API. Typically, retrieving the correct information from these legacy systems is done in a horrendous fashion – screen scraping! This is the art of emulating a screen based interface that was intended to display information to a human user. By sending the appropriate command sequences (usually keystrokes) to the system information is returned. Software libraries are used to then extract information from the ''virtual'' screen and process it as necessary. Obviously this approach is not recommended, but the reality is

[5] `http://www.cselt.stet.it/fipa.htm`

that the majority of cases where interaction with legacy systems is necessary there is no API and the development of an API cannot be justified.

Agents that interact with web-based systems are not really much better than screen-scrapers. These systems have to make do with downloading the html source of a web page and performing some parsing of the source to extract the necessary information. There are efforts to try to organise web based information in a more "system friendly" manner, such as the XML[6] standard, and research efforts like SHOE that provide meta-information about web-pages to allow information to be extracted in a cleaner fashion.

With any open system of this kind the definition of the interface includes the problem of how we unambiguously define the contents of the information that is to be passed between systems. Take, for example, a train timetable service that returns the time of the next train between two stations. If this is to be made completely open, any calling system would need to know how a station was defined – this could be by a unique station identifier; its latitude/longitude; its name and address; its post or zip code. The way this piece of information is defined changes how it could be used in an open environment significantly. A unique station identifier would be of no use to some application that needed to know what hotels were near the station, whereas some kind of coordinate-based approach would be more useful. This choice of naming, or ontology problem is common to all open systems, and is being addressed by a number of groups. The Knowledge Interchange Format (KIF) developed as part of the Stanford Knowledge sharing effort was one method for specifying the intent of the information being passed, using a formal first order logic representation. More recent work has focused on the web by defining mark-up standards giving meaning to the content of web pages (XML). Not surprisingly, the problem of a common ontology is being addressed as part of the standards initiative for FIPA in 1998 via an ontology service which supports the use of reference ontologies.

## 3 Agent systems development

Like any good software system development, there needs to be some methodology for developing collaborative software agent systems. Just as Jacobsen (1992) or Booch (1994) have defined methods for developing object-oriented systems, we would expect to see the same for agent oriented systems. Unfortunately, as agents have not been as universally pervasive as objects, there is not the maturity in agent oriented design. There are methods and guidelines such as the early (Agent0) language and guidelines that came out of Esprit funded projects such as Archon and Imagine.

The development of any large heterogeneous distributed agent-based system is always going to be difficult. Not only are there the issues of standard distributed systems development such as deadlock, livelock, etc. to be dealt with, this is compounded with the individual agents being able to plan/schedule tasks autonomously, and not necessarily behave in the same way each time they accomplish some complex task. It is not surprising that many of the papers presented at PAAM96 and PAAM97 (PAAM96; PAAM97) focused on the architectures and composition of their systems. The CLAIMS framework in particular gives an excellent view of the links between agents and objects (Malkoun, 1997). With this in mind, there has been much recent work on the development of toolkits to aid in the development of these systems, such as dMars from AAII, SRI with their Open Agent Architecture (Martin, 1996, 1997), Boeings KAOS (Bradshaw, 1997) and, more recently, through BT with Zeus (Nwana, 1998). Zeus in particular provides a suite of development tools for monitoring, logging and debugging in these environments.

## 4 Agent competence

Many agents need to adapt in the environment where they operate. Much of the early personal agents work focussed on watching user behaviour in particular situations and from that learning

[6] Extensible Markup Language, `http://www.w3.org/XML/`

rules to automate common actions. OpenSesame! was the first true commercial agent to do this in the desktop environment, where rules were learnt to automate common user sequences. The work by Lashkari (1994) on mail filtering showed how a case-based reasoning system was used to learn rules to deal with incoming mail.

In a business environment, agents should be able to recognise changing process flow and adapt accordingly. Ideally, we would like a system of agents that dynamically reconfigure themselves to take into account the changing workload and focus of workload. For example, in network traffic management, a mobile agent approach described by Appleby (1994) showed how mobile agents could be generated or culled depending on the need to control the network at any particular time.

The machine learning community has been working for years on effective methods for abstracting general principles from examples. There are various learning methods that can be used depending on the application area. Sen gives a good overview of these as applied to software agents. If we accept that learning to predict appropriate actions from past histories will always be error prone – it is not possible to predict the appropriate outcome for a new situation given some history. There will always be some degree of error. It is therefore important that the practical use of these algorithms is used in situations where this degree of error is acceptable. They should not, therefore, be central to any mission-critical application given that they may not predict the right control given a particular situation. One outcome of this is that we see these systems being used in non-critical areas such as recommender systems for records and films (Firefly[7]) (Fisk, 1996) and technical articles (Pazzani & Billsus).

It is interesting to note that of the personal agents developed, the ones that have succeeded to date have been those that filter and prioritise information, whereas those developed to automate actions have not been successful.

## 5 Trust and privacy

One of the main characteristics of agents is that you can delegate tasks to them. Tasks might be to prioritise e-mail, find information on your behalf or negotiate for goods and services for you. To be able to do most of these tasks effectively, the agent needs a good model of the users interests, needs and preferences. This information is in most instances of a very private and personal nature – it is not something that you would openly share.

Many of the underlying privacy issues should hopefully be addressed by the cryptographic infrastructures being put in place, for example the use of public and private keys, together with some certification authority (Verisign[8]). These can provide basic services such as:

- encryption of data to ensure it can only be read by the intended recipient;
- authentication of the sender or receiver of commands or information;
- digital signatures to prove that the message has not been tampered with on route;
- timestamping to prove when it was received; and
- non-repudiation mechanisms to ensure that if a person or system received the message, they cannot deny it was received.

All of these will be necessary for any open communications infrastructure for any type of electronic commerce initiatives, or in general any business to business transactions. These will provide the basis for protecting information. Even within an organisation the majority of these mechanisms are still necessary. If we take the applications of workflow management discussed by O'Brien later in this journal, we have agents negotiating the use of resources by different departments. Obviously the use of these resources costs money so there is the need for the resources to be accounted for between the departments, especially if they are semi-autonomous cost centres within an organisation.

[7] http://www.firefly.com
[8] http://www.verisign.com

The information that is known about you and your preferences should only be shared in a very controlled manner. Initiatives such as the Open Profiling Standard[9] go some way towards this. The standard only allows information to be shared directly between two parties – a client (the user) and server of a web site (the product or service provider), and the user is made aware that information has been requested.

## 6 Social issues

The preceding sections have discussed a number of the technical issues that faces the development of agent based systems. These in some cases are secondary to the social and political issues discussed here.

Agents are applications that use the services provided by a number of information sources. In some cases these information sources may be under the ownership of the same team that is developing the agent. In most cases, this will not be the case, so agreement must be found between the service provider and application user for this agent based access to be allowed. Bargainfinder was one early agent that managed to get banned from a number of CD stores because its aims were possibly not beneficial to any particular store. More recent work that might alleviate this type of problem is discussed by Gutmann et al. later in this issue.

Businesses want to maximise utility and minimise cost. Collaboration between different parts of business process is the mainstay of distributed agent solutions as discussed by O'Brien later in this issue. Although this method may help maximise utility it implies there is flexibility to switch between suppliers, etc. at minimum cost. Unless there is some advantage for all concerned there will not be the motivation required to move towards an agent based solution, so no amount of good theory will make any difference to the practical use (or not) of this type of system.

It is vitally important that users trust their agents to work effectively on their behalf. The only way that this can be done is by building up a ''relationship of trust'' between the user and agent. This means that there must be a lot of work put into the interface side of the agent development so it is clear that the user understands what the system ''knows'' of the user, and how that knowledge will be used.

## 7 Agent standards

The Foundation for Intelligent Physical Agents (FIPA) was set up in 1996 as a standards body to standardise agent specific needs and held its first meeting to coincide with the first PAAM conference in London. The focus is on standards for three main areas:

1. Agent management – the need to identify and find agents (white and yellow pages services) and the need to define their various states and who they can interact with.
2. Agent–agent interaction – standards covering the interaction between agents at the highest level, the meaning of messages that pass between agents – orders, requests, commitments, etc.
3. Agent–software interfaces – agents, like any software have a need to interact with software applications.

Except for the second standard on agent–agent interaction, the agent management and agent/software interaction standards address many problems faced by all large scale distributed software systems.

## 8 Summary and conclusions

The construction of agent-based systems has improved greatly over the years, with the development of frameworks and infrastructures for distributed and collaborative agent based systems. The

[9] http://www.w3.org/Submission/1997/6/

World Wide Web has provided a great testbed for many information finding agents or ''bots'' and will be the lifeblood of software agents for the immediate future.

Many of the early promises of agents have still not been met – how many agents are there that learn personal needs and execute tasks autonomously? How many distributed agent systems are there that work on a daily basis in large enterprises? In the first case, the lack of applications stems from the simple fact that it is a very hard problem to develop a user model that is accurate enough to be used in a predictive basis – labelling it an agent does not make the problem any easier! In the second case, re-engineering business solutions to work in a peer-to-peer fashion is a mammoth undertaking. If it was a completely new business with no existing infrastructure then an agent based solution would be feasible. With an existing infrastructure in place, much more effort is needed to educate for the benefits of agents. This education is still ongoing and the hype does not help.

So, what chance software agents? They will make it, but we may not recognise them in the end!

**Acknowledgements**

Thanks to Paul O'Brien and Hyacinth Nwana for critically reviewing earlier drafts of this paper.

**References**

Appleby S and Steward S, 1994. ''Mobile software agents for control in telecommunications networks'' BT Technology Journal 12(2) pp. 104–113.

Ashish N and Knoblock C, 1997. ''Wrapper generation for semi-structured internet sources.'' Proc. Workshop on Management of Semi-structured Data, 16 May 1997, pp. 10–17. AT&T Labs.

Booch G, 1994. *Object Oriented Analysis and Design with Applications* (*2nd ed.*). Benjamin/Cummins.

Bradshaw JM, 1997. ''Kaos toward an industrial strength open agent architecture.'' In *Software Agents*, AAAI Press/MIT Press.

Fisk D, 1996. ''An appliction of social filtering to movie recommendation.'' *BT Technology Journal* **14**(4) 124.

Genersereth MR and Ketchpel SP, 1994. ''Software agents.'' *Comm. ACM* **37**(7) 48–53.

Jacobson I, 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.

Kushmerick N, 1997. ''Wrapper induction for information extraction.'' PhD Thesis, TR UW-CSE-97-11-04, Department of Computer Science & Engineering, University of Washington.

Lashkari Y, Metral M and Maes P, ''Collaborative interface agents'' Proc of National Conference on Artificial Intelligence August 94.

Maes P, 1994. ''Agents that reduce work and information overload.'' *Comm. ACM* **37**(7) 31–40.

Malkoun MT, 1997. ''CLAIMS: Cooperative Layered Agents for Integrating Manufacturing Systems.'' Proc. PAAM 97, pp 237–253.

Martin D, Cheyer A and Lee GL, 1996. ''Agent development tools for the open agent architecture.'' Proc. PAAM 96, pp 387–404.

Martin D, Oohoma H, Moran D and Cheyer A, 1997. ''Information brokering in an agent architecture.'' Proc. PAAM 97, pp 467–486.

Nwana HS, 1996. ''Software agents: An overview.'' *Knowledge Engineering Review* **11**(3) November.

Nwana HS, Ndumu DT and Lee LC, 1998. ''ZEUS: An advanced toolkit for engineering distributed multi-agent systems.'' Proc. PAAM 98 (to appear).

PAAM, 1996. Proceedings of ''The Practical Application of Agents and Multi-agent Systems'' Conference, Westminster Hall, London, April.

PAAM, 1997. Proceedings of ''The Practical Application of Agents and Multi-agent Systems'' Conference, Westminster Hall, London, April.

Pazzani M and Billsus D, 1997. ''Learning and revising user profiles: The identification of interesting web sites'' Machine Learning 27(3).