Research and Development Challenges for Agent-Based Systems

Authors: Divine Ndumu & Hyacinth Nwana

Approved By:

---------------------------------------------------------
--
Nader Azarmi

Authorised By:

---------------------------------------------------------
--
Nader Azarmi (Project Manager)

MIDAS

Intelligent Systems Research
Advanced Applications and Technology

---

1 Accepted for publication in: IEE/BCS Software Engineering Journal (1996): special issue on Agents Technology.
Abstract

The increasing sophistication of today’s information era poses certain challenges to traditional information technology (IT) systems. Agent-based software technology is rapidly evolving to meet the demands of this new information era. However, before agent-based solutions can be routinely and successfully exploited in real world problems, first, certain fundamental research and software engineering issues have to be addressed. In this paper, we discuss some of the key challenges for research and development of agent-based software systems. Our discussion is carried out from the twin viewpoints of fundamental research questions and software engineering issues.

1 Introduction

Today’s information society is becoming ever increasingly sophisticated, with some of the hallmarks of this sophistication including: a steady growth in the production and consumption of information, heterogeneous and distributed information environments, a diversity of information processing needs and products, and in general, a greater reliance on computer-based information processing systems to provide, not only **accurate** and **timely**, but also **value-added** solutions to complex commercial and industrial problems. This has led to a state of affairs where traditional IT systems are increasingly hard-pressed to meet the challenges of, for example, integrating information from several distributed and heterogeneous sources, and processing this information to facilitate decision-making, while reducing the information overload on users. Agent-based software technology is rapidly evolving to meet the demands of this new information era. Application domains in which agent solutions are being applied or investigated include workflow management, telecommunications network management, air-traffic control, business process re-engineering, data mining, information retrieval/management, electronic commerce, education, personal digital assistants, e-mail filtering, digital libraries, command and control, smart databases, and scheduling/diary management.

However, in order for agent-based systems to provide, routinely and successfully, real world solutions to the challenges posed by the current information age, first, certain fundamental research and software engineering issues have to be addressed. In this paper, we discuss some of the key challenges for research and development of agent-based software systems. Our discussion is carried out from the viewpoints of fundamental research questions and software engineering issues.

To underpin our discussion of the challenges for agents research in subsequent sections, first, in Section 2, we attempt to define the notion of a software agent, and present an abstract framework for characterising software agents. For concreteness and to enhance understanding, we also describe some mappings from the abstract framework to a number of agent classes. Section 3 utilises the abstraction as a vehicle for discussing the main challenges for agents research. In Section 4, we discuss the main software engineering challenges in the development of agent-based systems. Section 5 discusses some social problems which are likely to follow the large-scale deployment of agent-based technology; and Section 6 concludes the paper.
A Conceptual Framework for Characterising Software Agents

2.1 What is an agent?

Despite the rampant and ubiquitous use of the phrase ‘agent-based’ in describing much of current software technology, there is, as yet, no commonly agreed definition of exactly what an agent is, or more importantly, what it is not. In this paper, we adopt the loose definitions of an agent as

“a self-contained program capable of controlling its own decision-making and acting, based on its perception of its environment, in pursuit of one or more objectives” (Jennings & Wooldridge 1996);

or equivalently as

“a component of software and/or hardware which is capable of acting exactly in order to accomplish tasks on behalf of its user” (Nwana 1996).

The former authors argue that four key attributes determine agenthood:

- **autonomy**: the ability to function largely independent of human intervention;
- **social ability**: the ability to interact ‘intelligently’ and constructively with other agents and/or humans;
- **responsiveness**: the ability to perceive the ‘environment’ and respond in a timely fashion to events occurring in it;
- **proactiveness**: the ability to take the initiative whenever the situation demands.

Nwana (1996) presents a complementary view of agenthood, underpinned by the attributes of autonomy and co-operative ability, but also including **learning ability**, i.e. the ability to improve performance over time.

Essentially therefore, a software agent is any software entity to which one can ascribe similar attributes implicit in the everyday usage of phrases such as ‘travel agent’ or ‘estate agent’. That is, among other things, goal-oriented behaviour, knowledge of the problem-solving techniques for their domain, autonomous and proactive functioning on behalf of a customer, and the ability to learn from experience of dealing with a customer or a particular problem. This analogy highlights certain social issues between a user/customer and her agent; for example, that of **trust**! Customers rarely trust their estate agents, what more a software agent. In Section 4 we discuss in some detail these issues.

Software programs not possessing a majority of the attributes of autonomy, social ability, responsiveness, proactiveness, and learning ability are **not** agents. For example, software **daemons** such as disk compressors do not qualify as agents because while they are autonomous and responsive, they are not proactive and lack social or learning abilities. In the same vein, although most expert systems are largely autonomous, they do not cooperate or learn, hence are not agents.

To summarise, an agent is any entity possessing the previously mentioned attributes; and
whose behaviour is best described by attributing to it an intention, and describing its actions in terms of their contribution to the attainment of that intention.

2.2 A framework for characterising software agents

As Nwana (1996) argues, “agents exist in a truly multi-dimensional space” which makes difficult their formal typological classification. However, it is possible, and perhaps enlightening, to view conceptually, an agent as composed of three layers (Moulin & Chaib-draa 1996; Nwana 1992): a definition layer, an organisation layer and a cooperation/coordination layer (Fig. 1).

Fig. 1: An abstract context diagram of a generic agent

At the definition layer, the agent is defined as an autonomous rational entity, i.e. in terms of its reasoning and learning mechanisms, goals, resources, skills, beliefs, preferences etc. At the organisation layer it is defined with respect to its relationships with other agents, e.g. what agencies it belongs to, what roles it plays in these agencies, what other agents it is aware of, what abilities it knows those other agents possess, etc. Note that our organisation layer encompasses both Moulin & Chaib-draa’s (1996) notion of structural organisation, i.e. the

“pattern of information and control relationships between agents and the distribution of problem-solving capabilities among them” (p. 18),

and also during runtime their notion of an organisation as

“a set of agents with [consistent] mutual commitments ... ” (p. 18).

At the coordination layer the social abilities of the agent are specified, e.g. what coordination/negotiation techniques it knows. The other two layers in Fig. 1 are included for completeness sake; the communication layer handles the low-level details involved in inter-agent communication, while the application programmer’s interface (API) layer links the agent to the physical realisations of its resources and skills.

We believe that this three-layer abstraction not only provides a framework for characterising agent applications, but also a structure for reviewing the research and development needs of agent-based systems. First, however, we utilise the abstraction to characterise a number of agent-based applications.

---

2 best — as in the description most comprehensible to the average person.

3 An agency is a group of related agents, i.e. they share a common attribute, for example, they may belong to the same company. Agencies may be virtual or real. A virtual agency is a group of agents who share some sort of cooperation agreement.
Nwana (1996) argues for a practical classification of agent applications (based in some cases on what the agents are, and in others on the roles they perform), resulting in the following agent types:

**Collaborative Agents:** typically static, large and coarse-grained agents which 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 generally a major emphasis of their operation. To coordinate their activities, they may have to negotiate in order to reach mutually acceptable agreements.

**Interface Agents:** support and provide proactive assistance, typically to a user using a complex application program. Interface agents emphasise autonomy and learning in order to perform tasks for their owners. The key metaphor underlying interface agents is that of a personal assistant who is collaborating with the user in the same work environment (Maes 1994). Their cooperation with other agents, if any, is typically limited to asking for advice.

**Mobile Agents:** software processes capable of roaming wide area networks (WANs) such as the world wide web (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.

**Information Agents:** proactive, dynamic, adaptive and cooperative WWW information managers which perform the role of managing, manipulating or collating information from many distributed sources.

**Reactive Agents:** 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.

**Hybrid Agents:** agents whose constitution is a combination of two or more agent philosophies.

**Heterogeneous Agent Systems:** an integrated set-up of agents belonging to a number of different agent classes.

Using our three-layer abstraction of agent systems, we can readily see that a collaborative agent is essentially one with a rich specification of all three definition, organisation and coordination layers. For instance, with respect to the definition layer, typical collaborative agent systems, e.g. the Pleiades system (Sycara 1995) and GRATE/GRATE* (Jennings 1995), utilise means-end plan-based reasoning, and have a rich and explicit representation of the goals, resources, abilities, beliefs and preferences of the agent. Furthermore, with respect to the organisation layer, there is a rich representation of the agent’s relationships with other agents, and its beliefs concerning those other agents. With respect to the coordination layer, there is also a rich representation of the agent’s social abilities, i.e. sophisticated coordination and negotiation abilities. In contrast, for interface agents, for example the Calendar Agent (Kozierok & Maes 1993) and NewT (Maes 1994), the agent definition layer encodes only a learning mechanism, with a limited, if any, reasoning mechanism. The goals, resources, and abilities of the agent are represented implicitly and cannot be reasoned about; and there is
generally no representation of its beliefs or preferences. The organisation layer, generally, contains only knowledge relating to the preferences and/or beliefs of the user. Typically, interface agents possess no social abilities, hence no coordination layer.

Mobile agents, in our abstract framework, are simply agents which, in their definition layer possess some mechanism for moving around WANs. Also, in our abstraction, information agents are essentially similar to interface agents, although the more sophisticated ones such as Etzioni & Weld’s (1994) internet softbot possess more advanced reasoning and goal handling mechanisms than found in typical interface agents; i.e. a richer definition layer. Reactive agents, in our abstract viewpoint, are those which possess no coordination or organisation layers, and whose definition layer contains, principally, a highly stylised or algorithmic reasoning mechanism without explicit representation or deliberation about the goals and/or beliefs of the agent.

Essentially, our abstract framework allows us to view an agent as characterised principally by three orthogonal dimensions of definition, organisation and coordination; with any particular agent application simply seen as a vector which maps to a point on this 3-space. Note however, that each principal orthogonal dimension might itself be characterised by a number of orthogonal attributes; for example, in the definition layer, reasoning and learning abilities are orthogonal. Furthermore, along each principal axis, there is a continuous spectrum from an impoverished specification to a very rich one. Hybrid agents illustrate this argument; a deliberative/reactive hybrid agent is basically one falling somewhere midway along the definition axis. Fig. 2 below illustrates our characterisation of some of the previously mentioned agent classes described in Nwana (1996).

![Fig. 2: Characterisation of agent application classes](image)

With regard to agents employing stronger definitions of agenthood, for example, agents imbued with mentalistic attributes such as beliefs, desires and intentions (Rao & Geogeff 1995), agents with emotional attitudes (Bates 1994), whether or not an agent is versatile, lies knowingly, is benevolent, etc., we can see that these attributes simply emphasise specific aspects of the agent definition layer.

Having outlined our abstract view of an agent, in the following section we utilise the structure that it provides as a vehicle for discussing the challenges for agents research.
Specifically, we will address the issue by considering in turn the definition, organisation and coordination layers.

3 Research Challenges

3.1 The definition layer

Because agent technology is essentially a mechanism for applying results from artificial intelligence (AI) research to practical real world problems, agent applications inherit some of the difficult questions prevalent in AI research. For decades, the main research areas in classical AI have been automated reasoning and learning, and the associated problem of knowledge representation. However, despite some progress, these problems remain largely unsolved and have carried on to agent applications. We discuss below the key problems associated with reasoning and learning in agent-based systems.

3.1.1 Reasoning

The cornerstone of most of classical AI work has been the physical symbol system hypothesis, initially advanced in the seminal work of Newell & Simon (1972) on Human Problem Solving. The hypothesis holds that for a physical system to demonstrate intelligent behaviour, it should be a symbol manipulating system. That is, events external to the system are feed into it in some symbolic form, which it manipulates, leading to behaviour. As such, this hypothesis immediately generates two problems (Wooldridge & Jennings 1995):

- the transduction and knowledge representation problem of capturing and encoding external events into some symbolic form, and
- the reasoning problem of utilising the symbolic information to determine appropriate behaviour.

Research on the reasoning problem has largely been based on the hypothesis that rational decision making is essentially a process of means-end planning. That is, given a goal, a rational agent would adopt a course of action which reduces the difference between her current state and her desired goal state. While significant progress has been made in planning research (for a review of the planning literature, see Steel (1987) and Hendler et al. (1990)), the state-of-the-art of automated planning still leaves a lot to be desired, particularly in their use as the core reasoning component of industrial multi-agent systems. We mention a few of the key problems below.

- Slow execution times: Arguably, the major practical problem with classical AI planners is their slow execution times. This means, in effect, that in moderately dynamic and unpredictable environments, the world changes faster than a plan-based agent can decide on actions to take to achieve its goals and execute those actions. Thus, the agent is continually reacting to the world and makes little progress towards the attainment of its goals.

- Commonsense reasoning: This is perhaps the second most important practical shortcoming of plan-based agents. In most applications, commonsense reasoning involving notions such as time, space, causality, etc. is handled in an ad hoc application-specific manner. Indeed, the issue of commonsense reasoning has defied formal treatment in AI, and presents perhaps the major stumbling block to the development of reasonably smart agent systems. However, recent efforts such as the CYC project which aims to
“create a commonsense substrate for the next generation of software” (Guha & Lenat 1994) may provide a practical solution to this problem.

- **Multi-agent planning:** The slow response times of plan-based agents are worsened in multi-agent environments where multi-agent planning is required. In a cooperative setting, in addition to the communications overheads to ensure cooperation, agents have to reason about the goals, plans and beliefs of other agents, in order, for example, to minimise duplication of effort. Even in competitive and antagonistic scenarios, with minimal communications overheads, agents still need to reason about other agents’ goals, plans and beliefs; for example, in order to sabotage those plans.

- **Modal reasoning:** The need for agents to reason about their own and other agent’s beliefs, and in dynamic and unpredictable environments, to reason using modalities such as possibility and necessitation, requires a modal reasoning capability. Current modal reasoning formalisms still face a number of problems; particularly the logical omniscience problem wherein an agent believes all the logical consequences of its beliefs (which is not so in real life), and the problem with opacity of beliefs where variable bindings in one belief statement cannot be assumed to hold in other belief statements. Thus far, therefore, most modal reasoning techniques cannot be readily applied in practical agent systems.

- **Temporal reasoning:** Reasoning about time still presents a major challenge in AI research, and hence agent applications. However, such reasoning is necessary and pervasive in most industrial applications.

So far, we have considered the problem of reasoning in agent-based systems from a process-oriented perspective, i.e. that of manipulating knowledge to produce behaviour. However, the associated problem of representing and managing this knowledge is equally thorny. The main questions here are twofold:

What knowledge representation formalism best supports plan-based reasoning, and

How do we ensure consistency of our agent’s knowledge base, i.e. truth maintenance.

**Knowledge representation:** Ndumu et al. (1995) argue that in any non-trivial domain, plan-based reasoning requires at least three different knowledge structures: an action specification for representing domain actions, a meta-knowledge level for reasoning about the plan construction process itself, and an object-knowledge level for reasoning about domain objects and for deducing the logical consequences of planned actions. Managing these three structures in an agent application of any complexity is obviously a non-trivial task. In addition, from a software engineering perspective, a related and potentially more important and error-prone task, particularly in multi-agent systems built by different developers, is that of defining and maintaining a common ontology for enabling knowledge sharing between agents. An ontology is a description of the concepts and relationships that can exist in a domain (Gruber 1993). In most applications, the ontology is largely implicit, making such systems inflexible and difficult to extend. The ARPA knowledge sharing effort is attempting to address this problem.

**Truth maintenance:** This presents a real challenge in agent-based systems, particularly those operating in dynamic and/or multi-agent environments. The issues here are (i) identifying contradictions in an agent’s knowledge base and (ii) determining whether and how to eliminate identified contradictions. The problem of identification is computationally straightforward but expensive, involving the maintenance of a network of the dependency
relations between facts in the agent’s knowledge base (Doyle 19??). The second problem is more complex because it introduces pragmatic domain- and problem-specific considerations. Essentially, a derived conclusion based on a number of independent assumptions might contradict a new fact. Which of the assumptions (or perhaps even the new fact) should be retracted to maintain consistency? Furthermore, is it worth the trouble, especially if this means communicating the retraction to a number of other agents who might have already started constructing plans based on this assumption and will now have to revise those plans? Barbuceanu & Fox (1996) describe an interesting approach to distributed truth maintenance in multi-agent systems which includes the pragmatics of the situation by considering the credibility of the new information and the utility of performing the retraction. However, there still remains the problem of determining the credibility or utility values of items of information.

The slow response times and brittle nature of agents based on classical AI planning, and the many unresolved problems associated with the symbolic AI stance, generated in the mid-eighties a radical rethink of the symbol manipulation hypothesis. Researchers such as Brooks (1986) and Agre & Chapman (1987) challenged the need for symbolic reasoning systems, arguing that in order to build intelligent agents it is necessary to have representations grounded in the physical world, with the agent connected to the world via sensors and actuators. Such agents directly sense the world, and react to it in a mechanistic fashion requiring no deliberation or planning. The reactive approach results in more robust, fault-tolerant, flexible and adaptable agents; however, it also suffers a number of significant drawbacks (Nwana 1996):

- Firstly, it is not clear how to design such agents so that the intended behaviour emerges as the agent reacts with its environment. Since it is not possible to tell the agent how to achieve some goal, “one has to find a ‘dynamics’, ... involving the system and the environment which will converge towards the desired goal” (Maes 1991). This is not only time-consuming, but also ad hoc.

- Secondly, as Maes (1991) notes, reactive agents have some important limitations precisely because “of their lack of explicit goals and goal-handling capabilities”, requiring the designers of such systems to precompile or hard-wire the action selections. Hence, while a planning approach leaves much to the agent, the reactive approach leaves much to the designers.

- Finally, how are such agents extended, scaled up or debugged? What happens if the ‘environment’ is changed?

Because of the relative merits of the reactive and deliberative philosophies, attempts have been made to combine both methods in a common reasoning framework, resulting in hybrid agent systems; e.g. InteRRap (Muller et al. 1995). While hybrid agents offer a number of practical advantages; however, they still suffer from similar theoretical problems as pure reactive agents. Perhaps, what is required, is a deliberative agent capable of reasoning about its own planning process, which in dynamic environments evaluates the trade-off between the time-consuming process of constructing an elaborate plan or utilising a pre-compiled plan (which is not guaranteed to work in that situation, but has a reasonable chance of success). This, however, leaves open the problem of determining appropriate trade-off functions for different scenarios. Perhaps this is best done by on-line learning from experience.

3.1.2 Learning
AI learning systems have enjoyed considerably more success than reasoning systems, which is why interface agents which emphasise learning over reasoning are generally more robust applications than collaborative agents. However, as with reasoning, in learning, the deliberative versus reactive dichotomy applies. Classical AI learning approaches are based on rule induction from examples (Quillian 19?9); and while relatively successful, suffer similar problems as deliberative reasoning systems: slow learning times, brittle, and inflexible. Recently, statistical and reactive learning methods such as neural networks (Rumelhart & McCleland 1986), evolutionary algorithms (Back 1996) and genetic algorithms (Whitley & Vose 1995) are becoming increasingly popular in agent applications because of their speed, robustness and generalisation ability. We identify the following challenges for learning in agent-based systems:

- What are the appropriate learning mechanisms for different types of problems and why?
- What are the effects of the various learning mechanisms on the responsiveness of agents?
- Would learning not lead to instability (especially in a multi-agent set-up)?
- How do you ensure an agent does not spend much of its time learning, instead of participating in its set-up?

With respect to interface agents in particular, as Mitchell et al. (1994) note

“...it remains to be demonstrated that knowledge learned by [such] systems ... can be used to significantly reduce their users’ workload”.

3.2 The organisation layer

The organisation layer, in our abstract agent framework, stores knowledge about an agent’s relationships with and beliefs about other agents. In the research literature, this layer has received significantly less attention than other areas. However, with regards to the goals of agents research, i.e. support for distributed and/or collaborative problem solving, the appropriate type and use of organisation layer knowledge goes a long way towards increasing the ‘intelligence’ of an agent. For example, for an interface agent say, collaborating with a user on some problem, the user’s perception of the value of the agent’s contribution to the problem solving effort depends on how well the agent can deduce the user’s goals, beliefs, and preferences, and respond in ways which respect the user’s beliefs while advancing her goals. To do this, the agent needs to maintain appropriate models of the user’s goals, beliefs, and preferences. We see the main research questions in this area as:

- Determining what organisational knowledge is required to support distributed and/or collaborative problem solving for different domains; and how this knowledge affects, supports and is affected by the different problem solving techniques that can be employed by agents. How can the knowledge be used to minimise problem solving effort? For example, in a multi-agent set-up, a superior-subordinate relationship between a pair of agents might suggest a client-server coordination strategy during problem solving, or alternatively a peer-to-peer relationship might suggest a contract-net and/or negotiated coordination strategy. What other structural and/or dynamic relationships can exist between agents; and how might these dictate appropriate coordination/negotiation techniques? For instance, Nwana (1994a) suggest a negotiation technique, gang-up negotiation, in which co-workers gang-up to persuade a least-busy colleague to accept a
high priority task. This techniques necessarily exploits the co-worker relationship between the negotiating agents.

- How is new organisational knowledge generated during problem solving; i.e. how are new beliefs about other agents generated from old beliefs and the current problem solving process?
- How does the evolution of the organisational knowledge of a community of agents affect their global evolution? Under what conditions does belief revision lead to instability or better global performance. Also, of particular importance to socio-economic organisation science, what are the effects on organisational performance of wrong beliefs; i.e. how do misconceptions about other agent’s beliefs, abilities and preferences affect group effort.

We believe research on the organisation layer could benefit significantly from input from other AI disciplines such as user modelling (this is especially true for interface agents research), human-computer interaction and collaborative work, and from social science disciplines such as organisational theory and decision making (e.g. see Gasser (1991)).

3.3 The coordination layer

In our abstract agent framework, the social abilities of an agent are defined in the coordination layer. Such abilities are particularly useful, and contribute significantly to the perceived ‘intelligence’ of the agent application, in collaborative multi-agent environments and/or when an agent needs to collaborate with a user. We concentrate here on the collaborative multi-agent scenario. Social abilities are necessary in collaborative multi-agent scenarios to ensure coherent behaviour of the community of agents, i.e. the community as a whole acts in a purposeful manner. In any non-trivial domain, in order to achieve coherence the individual group members have to coordinate their activity. Coordination is desirable for a number of reasons (Jennings 1996; Nwana 1994b; Nwana et al. 1996):

- preventing anarchy or chaos which is an inherent property of under-constrained complex non-linear and multi-dimensional systems;
- meeting global constraints which cannot be met by any one agent acting in isolation;
- maximising utility through sharing of distributed expertise, resources or information;
- minimising negative inter-dependencies between individual agent’s actions;
- and, in general, improving group efficiency by working as a team.

The research literature on coordination can be broadly classified into the following categories (Nwana et al. 1996): organisational structuring, contracting, multi-agent planning and negotiation.

In organisational structuring the role relationships between agents, e.g. superior-subordinate relationships, are used to drive the coordination process. This typically leads to the classic master-slave or client-server coordination technique, in which a master delegates tasks to a number of slaves. Here, it is the master’s responsibility to ensure there are no negative interactions between the activities of the slaves. This approach negates the benefits of the distributed problem solving abilities of the multi-agent scenario by concentrating control with master agents. However, it does reflect the type of problem solving that goes on in many real world organisations.
In contracting, typified by the classic *contract net protocol* (Smith 1980), a manager agent with a task to perform puts it out to tender to a number of contractor agents. The contractors willing to perform the task return bids to the manager, which get evaluated by the manager according to some criteria, and the contractor with the best bid gets allocated the task. Again, it is the manager’s responsibility to ensure there are no negative interactions between contractor agent’s activities. However, the approach naturally allows for load balancing in a community since agent’s decide on when to bid for tasks.

The multi-agent planning approach involves the construction of a plan with contributions from all the agents in the community who can contribute to the plan. Redundancy and conflicts between the agents’ activities are removed during the planning process. Typically, the multi-agent planning approach takes two forms: centralised planning, in which one agent manages the evolution of the plan and is responsible for conflict resolution; and decentralised planning wherein no one agent is responsible for the plan. The planning approach, particularly distributed planning carries a significant communications overhead, since agents’ have to communicate to one another their goals, expectations, beliefs, etc.

The final coordination technique, negotiation, or “the communication process of a group of agents in order to reach a mutually acceptable agreement on some matter” (Bussman & Muller 1992) is such a broad research area that we shall not attempt even a brief review here (see Nwana *et al.* (1996) for a gentle introduction to the subject). However, analysis of the literature suggests the following problems (Nwana *et al.* (1996):

- It is not clear when, where, how and why various coordination and negotiation strategies are used in various applications, hence, practical conclusions regarding their scope and applicability cannot be made.

- Most coordination and/or negotiation strategies do not involve any complex meta-reasoning required of most real domains. For example, few take into consideration the transient nature of agents’ goals, beliefs, etc.

We believe the key research challenges here are

- identification of coordination and/or negotiation strategies, and

- characterisation of these strategies.

Our contention is that different strategies more suited to different scenarios can be identified, and their range of applicability clearly delimited.

**4 Software Engineering Challenges**

The development of any new technology is typically characterised by a number of phases. In the first phase, there is generally a flurry of activity to identify and/or define methods and techniques for dealing with different classes of problems. This phase invariably involves a lot of *ad-hoc* application-specific designs and “re-invention of the wheel”, because in their drive to provide “point solutions to point problems”, researchers duplicate the efforts of other researchers. This is not necessarily a bad thing, since any duplication or redundancy confers greater confidence in the method or technology, and the flurry of activity results in a thorough exploration and hence a better understanding of the capabilities of the technology.

The second phase typically involves *consolidation, partitioning and methodological*
rationalisation. By consolidation, we mean that defects inherent in a particular method or facet of the technology have been weeded out, with these methods graduating into “tried-and-tested” techniques. Partitioning involves a decomposition of the problem space into well-understood classes, for which particular techniques of the technology are deemed most appropriate, i.e. an identification of which methods are most suitable for what problems and why. Following consolidation and partitioning is the development of structured methodological approaches for using the technology. A characteristic of the second phase is the graduation of the technology from a research tool into an enabling technology; that is, one where a user spends less time worrying about the intricacies of the technology, and spends the majority of her time trying to accurately capture the features of her problem. Another characteristic of this phase is the development of generic application-independent platforms to aid users of the technology. That is, the creation of solution libraries of user-customisable components, and an environment where a user can quickly apply the technology on her problem. In essence, use of the technology graduates from a scientific discipline in the first phase, where only experts in the particular technological domain can apply it; into an engineering disciple in the second phase where it can be employed by anyone in the solution of their problem. Additional advantages of a generic platform are support for code reuse and hence reduced development times for application developers, better engineered solutions, and a test-bed for experimentation to determine, for instance, optimal strategies for a particular problem.

The third and final phase generally comprises routine usage of the technology and its standardisation. Standardisation involves national and/or international agreements on the components of the technology and their usage.

The development of expert systems technology provides an excellent example of these developmental stages. Initially, expert systems were developed entirely from scratch (including the inference engine, knowledge base and user interface); latter, the knowledge contained in completed expert systems were removed to provide an empty expert system, or expert system shell (Shortiffe 1976), for use by others in developing their own systems. This has since been followed by the development of more sophisticated user-interfaces, solution libraries for different generic problem types, etc. Sadly, there has been little standardisation of expert systems technologies.

While conceding that there are still a number of theoretical issues which need to be resolved in agents research, we believe that agents technology has matured to the point where its main software engineering goal should be the development of generic platforms for engineering agent-based applications. The two words ‘generic’ and ‘engineering’ are especially significant. The platforms should be ‘generic’ in the sense that they provide structured methodological approaches for constructing any of the agent types discussed earlier. Engineering is used here in the sense that the development of an agent application using such a platform should follow a typical software engineering product development life-cycle: an analysis of the domain of application and functional requirements of the product, design of the product by selecting appropriate tools, components, and techniques from an established library, methodological construction of the product given the design, and evaluation of the finished product. Such platforms should provide the following benefits:

- An extension of the application areas of agents technology.
- Empirical evidence on the stability, scalability and performance of individual agents and agent communities.
• Progress towards the validation and verification of agent-based applications.

However, before such a platform can be developed, engineering solutions have to be devised for the following problems:

1. **Methodology:** What are the appropriate design methodologies for constructing the different types of agent systems for different application domains. Resolution of this question involves addressing a number of other questions including

   • **architectures:** what architectures are best suited for what problems and why?
   
   • **languages:** what are the desirable characteristics of implementation languages for the different types of agent systems? What are the necessary attributes of inter-agent communication languages and suitable support environments for inter-agent communication. For example, how can application- and platform-independent agent directory, brokering and nameserver services be implemented.

2. **The legacy software problem:** How should legacy software be integrated with agent-based applications in heterogeneous agent systems? The issue here is one of communication, i.e. enabling the legacy program to communicate with other agents. Genesereth & Ketchpel (1994) suggest three approaches for tackling this problem: legacy software rewrite — a costly approach, the **transducer** approach, and the **wrapper** technique. A transducer is a separate piece of software which acts as an interpreter between the inter-agent communication language and the legacy software’s native communication protocol. This approach is favoured in situations where the legacy code may be too delicate to tamper with or is unavailable. In the **wrapper** technique, some code is “injected” into the legacy program in order to allow it communicate in the agent communication language. The wrapper can access directly and modify the legacy program’s data structures. This is a more interventionist approach, but offers greater efficiency than the transduction approach.

3. **Mobile agents and remote programming:** Mobile agents introduce another dimension of software engineering issues. Wayner (1995) summarises the main challenges as

   • **Transportation:** how does an agent pack up and move from place to place?
   
   • **Authentication:** how do you ensure the agent is who it says it is, and is representing who it claims to be representing?
   
   • **Secrecy:** how do you ensure that your agents maintain your privacy? How do ensure others do not read your personal agent and execute it for their own gains? How do ensure your agent is not killed and its contents ‘core-dumped’?
   
   • **Security:** how do you protect against viruses? How do you prevent an incoming agent from entering an endless loop and consuming all the CPU cycles?
   
   • **Cash:** how will the agent pay for services? How do you ensure that it does not run amok and run up an outrageous bill on your behalf?

Some of these challenges are already being addressed successfully in development environments like the Telescript Development Environment (URL1), and using various techniques including the following: using ASCII-encoded, Safe-Tcl scripts or MIME-
compatible e-mail messages for transportation; using public- and private-key digital signature technology for authentication, cash and secrecy; and providing limited languages that will not allow an agent to write to memory, say, for security.

4. **Visualising and debugging multi-agent systems:** This is most important in collaborative agent applications that employ asynchronous inter-agent communications protocols. First, because of the number of agents involved, which are likely to be involved in different activities, it may prove difficult to give an application developer or user a clear picture of the community at any point in time or over a time period. Second, because of asynchronous communication, which is necessary to ensure scalability, messages from one agent to another are not guaranteed to reach their intended recipients; thus, the visualisation or debugging effort is even more complicated.

5 **Future Social Challenges**

In the preceding sections of this paper, we discussed the technical issues involved in agents research and development. However, in addition to these issues, there are also a range of social and ethical problems which will become increasingly important following the large scale fielding of agent technology, which society would have to deal with through various legislation. They include

- **Privacy:** how do you ensure your agents maintain your privacy when acting on your behalf?

- **Responsibility:** when you relinquish some of your responsibility to software agents be aware of the authority that is being transferred to them. How would you like to come home after a long hard day to find you are the proud owner of a used car negotiated and bought for, courtesy of one of your software agents?

- **Legal issues:** imagine your agent, which you bought off-the-shelf and customised, offers some bad advice to other peer agents, resulting in liabilities to other people — who is responsible? The company who wrote the agent? You who customised it? Both?

- **Ethical issues:** already, Eichmann (1994) and Etzioni & Weld (1994) are concerned enough about the ethics of software agents that they have proposed etiquettes for information service and user agents as they gather information on the WWW. Some of the conventions they propose are:

  - Agents must identify themselves;
  - They must moderate the pace and frequency of their requests to some server;
  - They must limit their searches to appropriate servers;
  - They must share information with others;
  - They must respect the authority placed on them by server operators;
  - Their services must be accurate and up-to-date;
  - They should not destructively alter the world;
  - They should leave the world as they found it;
  - They should limit their consumption of scarce resources;
They should not allow client actions with unanticipated results.

Most of these social problems, however, are not likely to be critical in the short-term development of agent applications, but will need to be addressed soon, in order to guarantee the future of agents technology.

6 Concluding Remarks

Proper use of a little knowledge goes a long way.

Software agents technology provides an excellent means for the down-streaming of results from more than two decades of artificial intelligence research towards solutions to problems created by the complexity of today’s information era. However, in order for agent-based solutions to be routinely and successfully exploited in real applications, a number of research and software engineering issues have to be addressed. In this paper, we have discussed some of the key challenges for the research and development of agent-based systems. We hope it provides a useful introduction to developers of agent technology. Nwana (1996) also includes a complementary alternative rendition of some of the challenges for agent-based applications.

Acknowledgements

This work was funded by BT Laboratories. The authors would like to thank the member of the Intelligent Systems Unit, BT Labs, who provided feedback which has improved the quality of this paper; however, we bear all responsibility for any misunderstandings or errors in the paper.

References


Doyle (19??)


Quillian (19??)


URL1 http://www.genmagic.com/

