Autonomous Agents and Multi-Agent Systems, 1, 275–306 (1998)

A Roadmap of Agent Research and Development

NICHOLAS R. JENNINGS n.r.jennings@qmw.ac.uk
Department of Electronic Engineering, Queen Mary and Westfield College, London E1 4NS, UK

KATIA SYCARA katia.sycara@cs.cmu.edu
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. 15213, USA

MICHAEL WOOLDRIDGE m.j.wooldridge@qmw.ac.uk
Department of Electronic Engineering, Queen Mary and Westfield College, London E1 4NS, UK

Received November 25, 1995; Revised March 30, 1996

Editor: ???

Abstract. This paper provides an overview of research and development activities in the field of autonomous agents and multi-agent systems. It aims to identify key concepts and applications, and to indicate how they relate to one-another. Some historical context to the field of agent-based computing is given, and contemporary research directions are presented. Finally, a range of open issues and future challenges are highlighted.

Keywords: autonomous agents, multi-agent systems, history

1. Introduction

Autonomous agents and multi-agent systems represent a new way of analysing, designing, and implementing complex software systems. The agent-based view offers a powerful repertoire of tools, techniques, and metaphors that have the potential to considerably improve the way in which people conceptualise and implement many types of software. Agents are being used in an increasingly wide variety of applications — ranging from comparatively small systems such as personalised email filters to large, complex, mission critical systems such as air-traffic control. At first sight, it may appear that such extremely different types of system can have little in common. And yet this is not the case: in both, the key abstraction used is that of an *agent*. It is the naturalness and ease with which such a variety of applications can be characterised in terms of agents that leads researchers and developers to be so excited about the potential of the approach. Indeed, several observers feel that certain aspects of agents are being dangerously over-hyped, and that unless this stops soon, agents will suffer a similar backlash to that experienced by the Artificial Intelligence (AI) community in the 1980s [96, 156].

Given this degree of interest and level of activity, in what is a comparatively new and multi-disciplinary subject, it is not surprising that the field of agent-based computing can appear chaotic and incoherent. The purpose of this paper is therefore to try and impose some order and coherence. We thus aim to tease out the common threads that together make up the agent tapestry. Our purpose is *not* to provide a detailed review of the field, we leave this to others (for example, [154, 11, 74, 111, 158]). Rather than present an in-depth analysis and critique of the field, we instead briefly introduce the key issues, and indicate how they are inter-related. Where appropriate, references to more detailed treatments are provided.

Before we can embark on our discussion, we first have to define what we mean by such terms as "agent", "agent-based system" and "multi-agent system". Unfortunately, we immediately run into difficulties, as some key concepts in the field lack universally accepted definitions. In particular, there is no real agreement even on the core question of exactly what an agent is (see [52] for a discussion). Of course, this need not be a serious obstacle to progress, (the AI community has made progress without having a universally accepted definition of intelligence, for example). Nevertheless, we feel it is worth spending some time on the issue, otherwise the terms we use will come to lose all meaning. For us, then, an agent is a computer system, *situated* in some environment, that is capable of *flexible autonomous* action in order to meet its design objectives (this definition is adapted from [158]). There are thus three key concepts in our definition: *situatedness*, *autonomy*, and *flexibility*.

Situatedness, in this context, means that the agent receives sensory input from its environment and that it can perform actions which change the environment in some way. Examples of environments in which agents may be situated include the physical world or the Internet. Such situatedness may be contrasted with the notion of *disembodied* intelligence that is often found in expert systems. For example, MYCIN, the paradigm expert system [134], did not interact directly with any environment. It received information not via sensors, but through a user acting as a middle man. In the same way, it did not act on any environment, but rather it gave feedback or advice to a third party.

Autonomy is a difficult concept to pin down precisely, but we mean it simply in the sense that the system should be able to act without the direct intervention of humans (or other agents), and that it should have control over its own actions and internal state. Others use it in a stronger sense, to mean systems that are capable of learning from experience [125, p35].

Of course, situated, autonomous computer systems are not a new development. There are many examples of such systems in existence. Examples include:

- any process control system, which must monitor a real-world environment and perform actions to modify it as conditions change (typically in real time) — such systems range from the very simple (for example, thermostats) to the extremely complex (for example, nuclear reactor control systems);

- software daemons, which monitor a software environment and perform actions to modify the environment as conditions change — a simple example is the UNIX `xbiff` program, which monitors a user's incoming email and obtains their attention by displaying an icon when new, incoming email is detected.

While the above are certainly examples of situated, autonomous systems, we would not consider them to be agents since they are not capable of flexible action in order to meet their design objectives. By *flexible*, we mean that the system is [158]:

- *responsive*: agents should perceive their environment and respond in a timely fashion to changes that occur in it;

- *pro-active*: agents should not simply act in response to their environment, they should be able to exhibit opportunistic, goal-directed behaviour and take the initiative where appropriate;
- *social*: agents should be able to interact, when appropriate, with other artificial agents and humans in order to complete their own problem solving and to help others with their activities.

While other researchers emphasise different aspects of agency (including, for example, mobility or adaptability) we believe that these four properties are the essence of agent-hood. Naturally, some agents will have additional characteristics, and for certain types of applications, some attributes will be more important than others. However, we believe that it is the presence of all the attributes in a single software entity that provides the power of the agent paradigm and which distinguishes agent systems from related software paradigms — such as object-oriented systems, distributed systems, and expert systems (see [156] for a discussion).

With the basic building block notion of an agent in place, we can define more of our terminology. By an *agent-based* system, we mean one in which the key abstraction used is that of an agent. In principle, an agent-based system might be conceptualised in terms of agents, but implemented without any software structures corresponding to agents at all. We can draw a parallel with object-oriented software, where it is entirely possible to design a system in terms of objects, but to implement it without the use of an object-oriented software environment. But this would at best be unusual, and at worst, counter-productive. A similar situation exists with agent technology; we therefore expect an agent-based system to be both designed and implemented in terms of agents.

As we have defined it, an agent-based system may contain one or more agents. There are cases in which a single agent solution is appropriate. A good example, as we shall see later in this article, is the class of systems known as expert assistants, wherein an agent acts as an expert assistant to a user attempting to use a computer to carry out some task. However, the *multi-agent* case — where the system is designed and implemented as several interacting agents — is arguably more general and more interesting from a software engineering standpoint. Multi-agent systems are ideally suited to representing problems that have multiple problem solving methods, multiple perspectives and/or multiple problem solving entities. Such systems have the traditional advantages of distributed and concurrent problem solving, but have the additional advantage of sophisticated patterns of interactions. Examples of common types of interactions include: cooperation (working together towards a common aim); coordination (organising problem solving activity so that harmful interactions are avoided or beneficial interactions are exploited); and negotiation (coming to an agreement which is acceptable to all the parties involved). It is the flexibility and high-level nature of these interactions which distinguishes multi-agent systems from other forms of software and which provides the underlying power of the paradigm.

The remainder of this article is structured as follows. Section 2 focuses on individual agents, providing some background to the concepts involved, indicating the key issues that are being addressed, and highlighting some likely future directions for research and development. Section 3 presents a similar discussion for multi-agent systems. Section 4 draws together the two strands. It discusses some exemplar agent-based applications and provides some pointers to the likely future direction of applied agent work. Finally, section 5 presents some conclusions.

2. Autonomous Agents

The purpose of this section is to identify the various threads of work that have resulted in contemporary research and development activities in agent-based systems. We begin, in section 2.1, by identifying some key developments in the history and pre-history of the autonomous agents area, up to and including current systems. In section 2.2, we then identify some of the key issues and future research directions in autonomous agents.

2.1. History

Current interest in autonomous agents did not emerge from a vacuum. Researchers and developers from many different disciplines have been talking about closely related issues for some time. The main contributors are:

- artificial intelligence [125];
- object-oriented programming [10] and concurrent object-based systems [2, 4];
- human-computer interface design [97].

2.1.1. Artificial Intelligence. Undoubtedly the main contributor to the field of autonomous agents is artificial intelligence. Ultimately, AI is all about building intelligent artifacts, and if these artifacts sense and act in some environment, then they can be considered as agents [125]. Despite the fact that agency can thus be seen to be central to the study of AI, until the 1980s comparatively little effort within the AI community was directed to the study of intelligent agents. The primary reason for this apparently strange state of affairs was that AI researchers had historically tended to focus on the various different *components* of intelligent behaviour (learning, reasoning, problem solving, vision understanding and so on) in isolation. The expectation was that progress was more likely to be made with these aspects of intelligent behaviour if they were studied individually, and that the *synthesis* of these components to create an integrated agent would be straightforward. By the early 1970s, this assumption seems to have been implicit within most mainstream AI research. During this period, the area of research activity most closely connected with that of autonomous agents was AI planning [5].

AI planning research is the sub-field of AI that concerns itself with knowing *what to do*: what action to perform. Ultimately, an agent is just a system that performs actions in some environment, and so it is not surprising that AI planning research should be closely involved in the study of agents. The AI planning paradigm traces its origins to Newell and Simon's GPS system [108], but is most commonly associated with the STRIPS planning system ([45]) and its descendents (such as [23, 155]). A typical STRIPS-style planning system will have at least the following components:

- a symbolic model of the agent's *environment*, typically represented in some limited subset of first-order predicate logic;
- a symbolic specification of the actions available to the agent, typically represented in terms of PDA (pre-condition, delete, add) lists, which specify both the circumstances under which an action may be performed and the effects of that action;
- a planning algorithm, which takes as input the representation of the environment, a set of action specifications, and a representation of a goal state, and produces as output a plan — essentially, a program — which specifies how the agent can act so as to achieve the goal.

Thus, planning systems decide how to act from *first principles*. That is, in order to satisfy a goal, they first formulate an entirely new plan or program for that goal. A planning systems would thus continually execute a cycle of picking a goal ϕ_1, generating a plan π for ϕ_1, executing π, picking a new goal ϕ_2, and so on. Crucially, such planning is based entirely around *symbolic representations and reasoning*.

Attention in AI planning research during the 1970s and early 1980s focussed primarily on the representations required for actions, and the planning algorithms themselves. Of particular concern was the demonstrated efficiency of the planning algorithm. With simulated micro-world examples (such as the well-known blocks world), STRIPS-style planning algorithms appear to give reasonable performance. However, it was rapidly discovered that such techniques do not scale to realistic scenarios. In particular, such algorithms were predicated on the assumption of *calculative rationality* [126]. The calculative rationality assumption may be informally defined as follows. Suppose we have some agent f, which will accept an observation of the world as input, do some computation, and sometime later generate an action output. Then f is said to enjoy the property of calculative rationality if the action it gives as output would be optimal if performed at the time f began its decision-making. Algorithms that have the property of calculative rationality by this definition will be guaranteed to make the best decision possible — but they will not necessarily make it in time to be of any use. Calculative rationality tends to arise in circumstances where the decision about which action to perform is made by an unconstrained search over the space of all possible decisions. The size of such search spaces are inherently exponential in the complexity of the task to be solved. As a consequence, search-based techniques tend to be impractical if results are required in any fixed time bound. Building on the burgeoning area of algorithmic complexity analysis that emerged in the late 1960s and 1970s, a number of theoretical results appeared in the 1980s which indicated that first-principles planning is not a viable option for agents that operate in such time-constrained environments. The best known of these results is due to David Chapman, who demonstrated that in many circumstances, first-principles planning is *undecidable* [23]. So, building *reactive* agents, that can respond to changes in their environment in time for these responses to be useful, is not likely to be possible using first-principles planning techniques.

The apparent failure of early AI planning techniques to scale up to real-world problems, together with the complexity results of Chapman and others, led many researchers to question the viability of symbolic reasoning approaches to planning in particular and AI in general. The problem of deploying such algorithms in real-world systems also prompted researchers to turn to the somewhat neglected issue of agent design. During the mid 1980s, an increasing number of these researchers began to question the assumptions upon which traditional symbolic AI approaches to agency are based. In particular, some researchers began to express grave reservations about whether symbolic AI, and in particular the *logicist* tradition in symbolic AI, was ultimately viable. Arguably the best-known of these critics was Rodney Brooks, who in a series of papers presented a number of objections to the symbolic AI model, and sketched out an alternative research program, which has been variously known as *behavioural AI*, *reactive AI*, or *situated AI* [15, 17, 16]. In these papers, Brooks emphasised several aspects of intelligent behaviour that he suggested were neglected by traditional approaches to agency and AI. In particular, he suggested that intelligent, rational behaviour is not an attribute of disembodied systems like theorem provers or traditional expert systems like MYCIN, but rather that intelligence is a product of the *interaction* between an agent and its environment. In addition, Brooks emphasised the view that intelligent behaviour *emerges* from the interaction of various simpler behaviours.

As part of his research program, Brooks developed the *subsumption architecture*, an agent control architecture that employed no symbolic representations or reasoning at all. Broadly speaking, a subsumption architecture agent is a collection of *task accomplishing behaviours*. Each behaviour is a finite state machine that continually maps perceptual input to action output. In some implemented versions of the subsumption architecture, this mapping is achieved via "situation $\rightarrow$ action" rules, which simply determine an action to perform on the basis of the agent's current state. However, in Brooks' implementations, behaviours were somewhat more sophisticated than this, for example allowing for feedback from previous decisions. The main point is that these behaviours do no symbolic reasoning (and no search). While each behaviour is generating suggestions with respect to which action to perform, the *overall* decision about which action to perform is determined by interactions between the behaviours. Behaviours can interact in several ways. For example, one behaviour can "suppress" the output of another. Typically, the behaviours are organised into a layered hierarchy, with lower layers representing less abstract behaviours (e.g., obstacle avoidance in physically embodied agents), and higher layers representing more abstract behaviours. Developing agents that exhibit coherent overall behaviour is a process of carefully developing and experimenting with new behaviours, usually by placing the agent in its environment and observing the results.

Despite its apparent simplicity, the subsumption architecture has nevertheless been demonstrated in several impressive applications (see, e.g., [137]). However, there are also a number of disadvantages with the subsumption architecture and its relatives:

- If agents do not employ models of their environment, then they must have sufficient information available in their *local* environment for them to determine an acceptable action.
- Since purely reactive agents make decisions based on *local* information, (i.e., information about the agent's *current* state), it is difficult to see how such decision making could take into account *non-local* information — it must inherently take a "short term" view.
- It is difficult to see how purely reactive agents can be designed that *learn* from experience, and improve their performance over time.

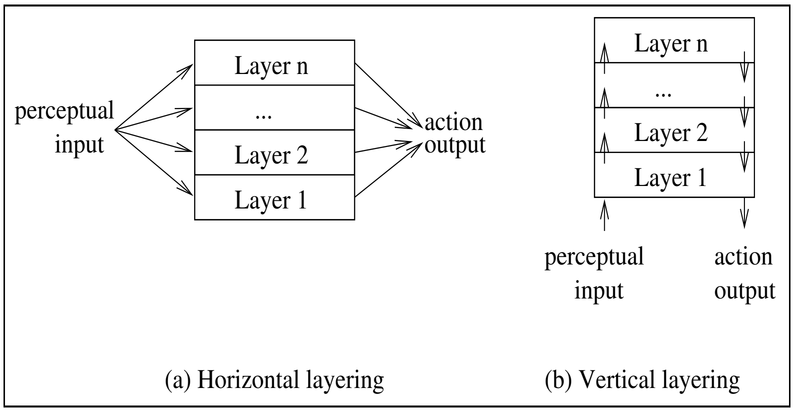

Figure 1. Layered Agent Architectures

- A major selling point of purely reactive systems is that overall behaviour *emerges* from the interaction of the component behaviours when the agent is placed in its environment. But the very term "emerges" suggests that the relationship between individual behaviours, environment, and overall behaviour is not understandable. This necessarily makes it very hard to *engineer* agents to fulfill specific tasks. Ultimately, there is no principled *methodology* for building such agents: one must use a laborious process of experimentation, trial, and error to engineer an agent.
- While effective agents can be generated with small numbers of behaviours (typically less that ten layers), it is *much* harder to build agents that contain many layers. The dynamics of the interactions between the different behaviours become too complex to understand.

By the early 1990s, most researchers accepted that reactive architectures are well-suited to certain domains and problems, but less well-suited to others. In fact, for most problems, neither a purely deliberative (e.g., first-principles planner) architecture nor a purely reactive architecture is appropriate. For such domains, an architecture is required that incorporates aspects of both. As a result, a number of researchers began to investigate *hybrid* architectures, which attempted to marry the best aspects of both deliberative and reactive approaches. Typically, these architectures were realised as a number of software *layers*. The layers may be arranged *vertically* (so that only one layer has access to the agent's sensors and effectors) or *horizontally* (so that all layers have access to sensor input and action output); see Figure 1.

As in the subsumption architecture, (see above), layers are arranged into a hierarchy, with different levels in the hierarchy dealing with information about the environment at different levels of abstraction. Most architectures find three layers sufficient. Thus at the lowest level in the hierarchy, there is typically a "reactive" layer, which makes decisions about what to do based on raw sensor input. Often, this layer is implemented using techniques rather similar to Brooks' subsumption architecture (thus it is itself implemented as a hierarchy of task accomplishing behaviours, where these behaviours are task-accomplishing finite state machines). The middle layer layer typically abstracts away from raw sensor input and deals with a *knowledge level* view of the agent's environment [107], typically making use of symbolic representations. The uppermost level of the architecture tends to deal with the *social* aspects of the environment — it has a *social knowledge level* view [80]. We thus typically find representations of other agents in this layer — their goals, beliefs, and so on. In order to produce the global behaviour of the agent, these layers interact with one-another; the specific way that the layers interact differs from architecture to architecture. In some approaches (such as TOURING MACHINES [43, 44]), each layer is itself constantly producing suggestions about what action to perform. In this case, mediation between these layers in order to ensure that the overall behaviour of the agent is coherent and consistent becomes an issue. In TOURING MACHINES this mediation is achieved by a *control subsystem* that determines which layer should have overall control of the agent. The control subsystem in TOURING MACHINES is implemented as a set of rules, which can refer to the actions proposed by each layer. A similar idea is used in INTERRAP [104, 103]. Another similar architecture for autonomous agents is 3T [8].

A final tradition in the area of agent architectures is that of *practical reasoning* agents [14]. Practical reasoning agents are those whose architecture is modelled on or inspired by a theory of practical reasoning in humans. By practical reasoning, we simply mean the kind of pragmatic reasoning that we use to decide what to do. Practical reasoning has long been an area of study by philosophers, who are interested in developing theories that can account for human behaviour. Typically, theories of practical reasoning make use of a *folk psychology*, whereby behaviour is understood by the attribution of *attitudes* such as beliefs, desires, intentions, and so on. Human behaviour can be thought of as arising through the interaction of such attitudes. Practical reasoning architectures are modelled on theories of such interactions. Probably the best-known and most influential type of practical reasoning architecture is the so-called *belief-desire-intention* (BDI) model [14, 57]. As the name indicates, BDI agents are characterised by a "mental state" with three components: beliefs, desires, and intentions. Intuitively, beliefs correspond to information that the agent has about its environment. Desires represent "options" available to the agent — different possible states of affairs that the agent may choose to commit to. Intentions represent states of affairs that the agent has chosen and has committed resources to. An agent's practical reasoning involves repeatedly updating beliefs from information in the environment, deciding what options are available, "filtering" these options to determine new intentions, and acting on the basis of these intentions. The philosophical foundations of the BDI model are to be found in Bratman's account of the role that intentions play in human practical reasoning [12]. A number of BDI agent systems have been implemented, the best-known of which is probably the *Procedural Reasoning System* (PRS) [57]. Researchers interested in practical reasoning architectures have developed a number of logical theories of BDI systems [120, 121]. Closely related to this work on practical reasoning agent architectures is Shoham's proposal for *agent-oriented programming*, a multi-agent programming model in which agents are explicitly programmed in terms of mentalistic notions such as belief and desire [133].

2.1.2. Object and Concurrent Object Systems. Object-oriented programmers often fail to see anything novel or new in the idea of agents. When one stops to consider the relative properties of agents and objects, this is perhaps not surprising. Objects are defined as computational entities that *encapsulate* some state, are able to perform actions, or *methods* on this state, and communicate by message passing.

While there are obvious similarities, there are also significant differences between agents and objects. The first is in the degree to which agents and objects are autonomous. Recall that the defining characteristic of object-oriented programming is the principle of encapsulation — the idea that objects can have control over their own internal state. In programming languages like JAVA, we can declare instance variables (and methods) to be `private`, meaning they are only accessible from within the object. (We can of course also declare them `public`, meaning that they can be accessed from anywhere, and indeed we must do this for methods so that they can be used by other objects. But the use of `public` instance variables is generally considered poor programming style.) In this way, an object can be thought of as exhibiting autonomy over its state: it has control over it. But an object does not exhibit control over it's *behaviour*. That is, if a method `m` is made available for other objects to invoke, then they can do so whenever they wish; the object has no control over whether or not that method is executed. Of course, an object *must* make methods available to other objects, or else we would be unable to build a system out of them. This is not normally an issue, because if we build a system, then we design the objects that go in it, and they can thus be assumed to share a "common goal". But in many types of multi-agent system, (in particular, those that contain agents built by different organisations or individuals), no such common goal can be assumed. It cannot be taken for granted that an agent i will execute an action (method) a just because another agent j wants it to — a may not be in the best interests of i. We thus do not think of agents as invoking methods upon one-another, but rather as *requesting* actions to be performed. If j requests i to perform a, then i may perform the action or it may not. The locus of control with respect to the decision about whether to execute an action is thus different in agent and object systems. In the object-oriented case, the decision lies with the object that invokes the method. In the agent case, the decision lies with the agent that receives the request. The distinction between objects and agents can be summarised in the following slogan: *Objects do it for free; agents do it for money*.

Note that there is nothing to stop us implementing agents using object-oriented techniques. For example, we can build some kind of decision making about whether to execute a method into the method itself, and in this way achieve a stronger kind of autonomy for our objects. However, the point is that autonomy of this kind is not a component of the basic object-oriented model.

The second important distinction between object and agent systems is with respect to the notion of flexible (reactive, pro-active, social) autonomous behaviour. The standard object model has nothing whatsoever to say about how to build systems that integrate these types of behaviour. Again, one could argue that we can build object-oriented programs that *do* integrate these types of behaviour. But this argument misses the point, which is that the standard object-oriented programming model has nothing to do with these types of behaviour.

The third important distinction between the standard object model and our view of agent systems is that agents are each considered to have their own thread of control — in the standard object model, there is a single thread of control in the system. Of course, a lot of work has recently been devoted to *concurrency* in object-oriented programming. For example, the JAVA language provides built-in constructs for multi-threaded programming. There are also many programming languages available (most of them admittedly prototypes) that were specifically designed to allow concurrent object-based programming [4]. But such languages do not capture the idea we have of agents as *autonomous* entities. Note, however, that *active objects* come quite close to our concept of autonomous agents — though not agents capable of flexible autonomous behaviour [10, p91].

2.1.3. Human-Computer Interfaces Currently, when we interact with a computer via a user interface, we are making use of an interaction paradigm known as *direct manipulation*. Put simply, this means that a computer program (a word processor, for example) will only do something if we explicitly tell it to. This makes for very one-way interaction. It would be desirable, therefore, to have computer programs that in certain circumstances could *take the initiative*, rather than wait for the user to spell out exactly what they wanted to do. This leads to the view of computer programs as *cooperating* with a user to achieve a task, rather than acting simply as servants. A program capable of taking the initiative in this way would in effect be operating as a semi-autonomous agent. Such agents are sometimes referred to as *expert assistants*, or more whimsically as *digital butlers*.

One of the key figures in the development of agent-based interfaces has been Nicholas Negroponte. His vision of agents at the interface was set out in *Being Digital* [106]:

> "The 'agent' answers the phone, recognizes the callers, disturbs you when appropriate, and may even tell a white lie on your behalf. The same agent is well trained in timing, versed in finding opportune moments, and respectful of idiosyncrasies. " (p150)

> "If you have somebody who knows you well and shares much of your information, that person can act on your behalf very effectively. If your secretary falls ill, it would make no difference if the temping agency could send you Albert Einstein. This issue is not about IQ. It is shared knowledge and the practice of using it in your best interests." (p151)

> "Like an army commander sending a scout ahead ... you will dispatch agents to collect information on your behalf. Agents will dispatch agents. The process multiplies. But [this process] started at the interface where you delegated your desires." (p158)

The main application of such agents to date has been in the area of information management systems, particularly email managers and active new readers [97] and active world-wide web browsers [94]. In section 4.1.2, we discuss such applications in more detail.

2.2. *Issues and Future Directions*

The area of agent architectures, particularly layered, or *hybrid* architectures, and practical reasoning architectures, continues to be an area of considerable research effort within the agent field. For example, there is ongoing work to investigate the appropriateness of various architectures for different environment types. It turns out to be quite hard to evaluate one agent architecture against another, although some suggestions have been made as to how this might be done in a neutral way [118].

Finally, if agent technology, of the kind described in this section, is to move from the research lab to the office of the everyday computer worker, then serious attention must be given to *development environments* and *programming languages* for such systems. To date, most architectures have been implemented in a rather *ad hoc* manner. Programming languages and tools for agents would present the developer with a layer of abstraction over such architectures. Shoham's AGENT0 is one attempt to build such a language [133], as is the CONGOLOG language described in [89], and the Concurrent METATEM programming language [47]. APRIL is another such language, which provides the developer with a set of software tools for implementing MAS [99].

3. Multi-Agent Systems

Traditionally, research into systems composed of multiple agents was carried out under the banner of *Distributed Artificial Intelligence* (DAI), and has historically been divided into two main camps [9]: *Distributed Problem Solving* (DPS) and *Multi-Agent Systems* (MAS). More recently, the term "multi-agent systems" has come to have a more general meaning, and is now used to refer to all types of systems composed of multiple (semi-)autonomous components.

Distributed problem solving (DPS) considers how a particular problem can be solved by a number of modules (nodes), which cooperate in dividing and sharing knowledge about the problem and its evolving solutions. In a pure DPS system, all interaction strategies are incorporated as an integral part of the system. In contrast, research in MAS is concerned with the behavior of a collection of possibly pre-existing autonomous agents aiming at solving a given problem. A MAS can be defined as a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver [39]. These problem solvers — agents — are autonomous and may be heterogeneous in nature. The characteristics of MAS are:

- each agent has incomplete information, or capabilities for solving the problem, thus each agent has a limited viewpoint;
- there is no global system control;
- data is decentralized; and
- computation is asynchronous.

Some reasons for the increasing interest in MAS research include: the ability to provide robustness and efficiency; the ability to allow inter-operation of existing legacy systems; and the ability to solve problems in which data, expertise, or control is distributed. Although MAS provide many potential advantages, they also face many difficult challenges. Below, we present problems inherent in the design and implementation of MAS (this list includes both problems first posed in [9] and some we have added):

1. How to formulate, describe, decompose, and allocate problems and synthesize results among a group of intelligent agents?
2. How to enable agents to communicate and interact? What communication languages and protocols to use? What and when to communicate?
3. How to ensure that agents act coherently in making decisions or taking action, accommodating the nonlocal effects of local decisions and avoiding harmful interactions?
4. How to enable individual agents to represent and reason about the actions, plans, and knowledge of other agents in order to coordinate with them? How to reason about the state of their coordinated process (e.g., initiation and completion)?
5. How to recognize and reconcile disparate viewpoints and conflicting intentions among a collection of agents trying to coordinate their actions?
6. How to effectively balance local computation and communication? More generally, how to manage allocation of limited resources?
7. How to avoid or mitigate harmful overall system behavior, such as chaotic or oscillatory behavior?
8. How to engineer and constrain practical MAS systems? How to design technology platforms and development methodologies for MAS?

Solutions to these problems are of course intertwined [54]. For example, different modeling schemes for an individual agent may constrain the range of effective coordination regimes; different procedures for communication and interaction have implications for behavioral coherence; different problem and task decompositions may yield different interactions.

From this backdrop, we provide some historical context for the field (section 3.1), discuss contemporary work in distributed problem solving (section 3.2) and multi-agent systems (section 3.3), and finally, we discuss some open issues (section 3.4).

3.1. History

In 1980 a group of AI researchers held the first DAI workshop at MIT to discuss issues concerning intelligent problem solving with systems consisting of multiple problem solvers. It was decided that Distributed AI was not concerned with low level parallelism issues, such as how to distribute processing over different machines, or how to parallelize centralized algorithms, but rather with issues of how intelligent problem solvers could coordinate effectively to solve problems. From these beginnings, the DAI field has grown into a major international research area.

3.1.1. Actors One of the first models of multi agent problem solving was the *actors* model [2, 3]. Actors were proposed as universal primitives of concurrent computation. Actors are self-contained, interactive autonomous components of a computing system that communicate by asynchronous message passing. The basic actor primitives are:

- *create*: creating an actor from a behavior description and a set of parameters, possibly including existing actors;
- *send*: sending a message to an actor;
- *become*: changing an actor's local state.

Actor models are a natural basis for many kinds of concurrent computation. However, as noted in [9] actor models, along with other DAI models, face the issue of coherence. The low-level granularity of actors also poses issues relating to the composition of actor behaviors in larger communities, and achievement of higher level performance goals with only local knowledge. These issues were addressed in [68] where an overview of *Open Systems Science* and its challenges were presented, and where an organizational architecture called ORG was proposed that included new features and extensions of the Actor model to support organizing large scale work.

3.1.2. Task Allocation through the Contract Net Protocol The issue of flexible allocation of tasks to multiple problem solvers (nodes) received attention early on in the history of DAI [33]. Davis and Smith's work resulted in the well-known *Contract Net Protocol*. In this protocol, agents can dynamically take two roles: *manager* or *contractor*. Given a task to perform, an agent first determines whether it can break it into subtasks that could be performed concurrently. It employs the Contract Net Protocol to announce the tasks that could be transferred, and requests bids from nodes that could perform any of these tasks. A node that receives a task announcement replies with a bid for that task, indicating how well it thinks it can perform the task. The contractor collects the bids and awards the task to the best bidder. Although the Contract Net was considered by Smith and Davis (as well as many subsequent DAI researchers) to be a *negotiation* technique, it is really a *coordination method for task allocation*. The protocol enables dynamic task allocation, allows agents to bid for multiple tasks at a time, and provides natural load balancing (busy agents need not bid). Its limitations are that it does not detect or resolve conflicts, the manager does not inform nodes whose bids have been refused, agents cannot refuse bids, there is no pre-emption in task execution (time critical tasks may not be attended to), and it is communication intensive. To rectify some of its shortcoming, a number of extensions to the basic protocol have been proposed, for example [128].

3.1.3. Some Early Applications

Air Traffic Control: Cammarata [21] studied cooperation strategies for resolving conflicts among plans of a group of agents. They applied these strategies to an air-traffic control domain, in which the aim is to enable each agent (aircraft) to construct a flight plan that will maintain a safe distance with each aircraft in its vicinity and satisfy additional constraints (such as reaching its destination with minimal fuel consumption). Agents involved in a potentially conflicting situation (e.g., aircraft becoming too close according to their current flight path) choose one of the agents involved in the conflict to resolve it. The chosen agent acts as a centralized planner to develop a multi-agent plan that specifies the conflict-free flight paths that the agents will follow. The decision of which agent will do the planning is based on different criteria, for example, most-informed agent, or more-constrained agent. The authors carried out experimental evaluations to compare plans made by agents that were chosen using different criteria.

The Distributed Vehicle Monitoring Task (DVMT): In this domain, a set of agents are distributed geographically, and each is capable of sensing some portion of an overall area to be monitored. As vehicles move through its sensed area, each agent detects characteristic sounds from those vehicles at discrete time intervals. By analyzing the combination of sounds heard from a particular location at a specific time, an agent can develop interpretations of what vehicles might have created these sounds. By analyzing temporal sequences of vehicle interpretations, and using knowledge about mobility constraints of different vehicles, the agent can generate tentative maps of vehicle movements in its area. By communicating tentative maps to one another, agents can obtain increased reliability and avoid redundant tracking in overlapping regions [38].

Blackboards: The DVMT, along with other early MAS applications use the *blackboard systems* for coordination. Put crudely, a blackboard is simply a shared data structure [40]. Agents can use a blackboard to communicate by simply writing on the data structure. Early DVMT work by Lesser and Corkill [30] used two blackboards, one for data and the other for agents' goals. In the MINDS project, Huhns et al also used two specialized blackboards [73]. The MINDS project was a distributed information retrieval system, in which agents shared both knowledge and tasks in order to cooperate in retrieving documents for users. Hayes-Roth proposed a more elaborate blackboard structure, with three interacting sub-agents for perception, control and reasoning [64].

3.2. Cooperative Multi-Agent Interactions

As interest increases in applications that use cooperative agents working towards a common goal, and as more agents are built that cooperate as teams, (such as in virtual training [143], Internet-based information integration [35], ROBOCUP robotic and synthetic soccer [149], and interactive entertainment [66]), so it becomes more important to understand the principles that underpin cooperation.

As discussed in section 2.1, planning for a single agent is a process of constructing a sequence of actions considering only goals, capabilities and environmental constraints. Planning in a MAS environment, on the other hand, considers in addition the constraints that the other agents' activities place on an agent's choice of actions, the constraints that an agent's commitments to others place on its own choice of actions and the unpredictable evolution of the world caused by other, un-modeled agents.

Most early work in DAI dealt with groups of agents pursuing common goals (e.g., [90, 38, 21, 91]). Agent interactions were guided by cooperation strategies meant to improve their collective performance. In this light, early work on distributed planning took the approach of complete planning before action. To produce a coherent plan, the agents must be able to recognize subgoal interactions and either avoid them or else resolve them. For instance, work by Georgeff [55] included a synchroniser agent to recognize and resolve such interactions. Other agents send this synchroniser their plan; the synchroniser examines plans for critical regions in which, for example, contention for resources could cause them to fail. The synchroniser then inserted synchronization messages (akin to operating systems semaphores) to ensure mutual exclusion. In the work by Cammarata on air traffic control (see section 3.1.3) the synchronizing agent was dynamically assigned according to different criteria, and could alter its plan to remove the interaction (avoid collision).

Another significant approach to resolving sub-problem interdependencies is the "Functionally Accurate Model (FA/C)" [91]. In the FA/C model, agents do not need to have all the necessary information locally to solve their sub-problems, but instead interact through the asynchronous, coroutine exchange of partial results. Starting with the FA/C model, a series of sophisticated distributed control schemes for agent coordination were developed, such as use of static meta-level information specified by an organizational structure, and the use of dynamic meta-level information developed in *Partial Global Planning* (PGP) [38].

Partial Global Planning is a flexible approach to coordination that does not assume any particular distribution of sub-problems, expertise or other resources, but instead allows nodes to coordinate themselves dynamically [38]. Agent interactions take the form of communicating plans and goals at an appropriate level of abstraction. These communications enable a receiving agent to form expectations about the future behavior of a sending agent, thus improving agent predictability and network coherence [38]. Since agents are cooperative, the recipient agent uses the information in the plan to adjust *its own* local planning appropriately, so that the common planning goals (and planning effectiveness criteria) are met. Besides their common PGP's, agents also have some common knowledge about how and when to use PGPs. Decker [34] addressed some of the limitations of the PGP by creating a generic PGP-based framework called TAEMS to handle issues of real-time (e.g., scheduling to deadlines) and meta-control (e.g., to obviate the need to do detailed planning at all possible node interactions).

Another research direction in cooperative multi-agent planning has been directed towards modeling teamwork explicitly. This is particularly helpful in dynamic environments, where team members may fail or where they may be presented with new opportunities. In such situations, it is necessary that teams monitor their performance and reorganize based on their current situation.

The joint intentions framework [93] is a natural extension to the practical reasoning agents paradigm discussed in section 2.1.1. It focuses on characterising a team's mental state, *called a joint intention* (see e.g., [77] for survey). A team jointly intends a team action if the team members are jointly committed to completing the team action, while mutually believing they were doing it. A joint commitment is defined as a joint persistent goal. To enter into a joint commitment, all team members must establish appropriate mutual beliefs and commitments. This is done through an exchange of request and confirm speech acts [28]. The commitment protocol synchronizes the team, in that all members simultaneously enter into a joint commitment towards a team task. In addition, all team members must consent, via confirmation, to the establishment of a joint commitment goal

— thus a joint commitment goal is not established if a team member refuses. In this case, negotiation could be used, though how this might be done remains an open issue.

The SharedPlan model [60, 61] is based on a different mental attitude: *intending that* an action be done [13]. "Intending that" concerns a group's joint activity or a collaborator's actions. The concept is defined via a set of axioms that guide a team mate to take action, or enter into communication that enables or facilitates its team mates to perform assigned tasks. COLLAGEN [122] is a prototype toolkit that has its origins in the SharedPlan model, and which has been applied to building a collaborative interface agent that helps with air travel arrangements. Jennings [79] presented a framework called *joint responsibility* based on a joint commitment to a team's joint goal and a joint recipe commitment to a common recipe. This model was implemented in the GRATE* system [78], and applied to the domain of electricity transport management.

Tambe [144] presents a model of teamwork called STEAM (*S*hell for *TEAM*work), based on enhancements to the Soar architecture [109], plus a set of about 300 domain independent Soar rules. Based on the teamwork operationalized in STEAM, three teams have been implemented, two that operate in a commercially available simulation for military training and a third in ROBOCUP synthetic soccer. STEAM uses a hybrid approach that combines joint intentions with partial SharedPlans.

3.3. Self-Interested Multi Agent Interactions

The notion of interactions among self-interested agents has been centered around *negotiation*. Negotiation is seen as a method for coordination and conflict resolution (e.g., resolving goal disparities in planning, resolving constraints in resource allocation, resolving task inconsistencies in determining organizational structure). Negotiation has also been used as a metaphor for communication of plan changes, task allocation, or centralized resolution of constraint violations. Hence, negotiation is almost as ill-defined as the notion of "agent".

We give here what we consider to be the main characteristics of negotiation, that are necessary for developing applications in the real world. These are: (a) the presence of some form of conflict that must be resolved in a decentralized manner, by (b) self-interested agents, under conditions of (c) bounded rationality, and (d) incomplete information. Furthermore, the agents communicate and iteratively exchange proposals and counter-proposals.

The PERSUADER system by Sycara [141, 140] and work by Rosenschein [123, 124] represent the first work by DAI researchers on negotiation among self-interested agents. The two approaches differ in their assumptions, motivations, and operationalization. The work of Rosenschein was based on game theory. Utility is the single issue that agents consider, and agents are assumed to be omniscient. Utility values for alternative outcomes are represented in a payoff matrix that is common knowledge to both parties in the negotiation. Each party reasons about and chooses the alternative that will maximize its utility. Despite the mathematical elegance of game theory, game theoretic models suffer from restrictive assumptions that limit their applicability to realistic problems[1]. Real world negotiations are conducted under uncertainty, involve multiple criteria rather than a single utility dimension, the utilities of the agents are not common knowledge but are instead private, and the agents are not omniscient.

The PERSUADER is an implemented system that operates in the domain of labor negotiation [139]. It involves three agents (a union, a company, and a mediator), and is inspired by human negotiation. It models the iterative exchange of proposals and counter-proposals in order for the parties to reach agreement. The negotiation involves multiple issues, such as wages, pensions, seniority, subcontracting, and so on. Each agent's multi-dimensional utility model is private (rather than common) knowledge. Belief revision to change the agents' utilities so that agreement can be reached is achieved via persuasive argumentation [141]. In addition, case-based learning techniques are also incorporated into the model.

Work by Kraus [86], focuses on the role of time in negotiation. Using a distributed mechanism, agents negotiate and can reach efficient agreements without delays. It is also shown that the individual approach of each agent towards the negotiation time affects (and may even determine) the final agreement that is reached.

As electronic commerce is rapidly becoming a reality, the need for negotiation techniques that take into consideration the complexities of the real world, such as incomplete information, multiple negotiation issues, negotiation deadlines, and the ability to break contracts will be critically needed. Work in non-binding contracts includes [127] where decommitment penalties were introduced into the Contract Net Protocol, and the ADEPT system, in which a penalty for contract violation was built into the negotiation agreement [82].

Another important aspect of successful interaction for self-interested agents is the ability to adapt behaviour to changing circumstances (see [138] for a survey). However, learning in a multi-agent environment is complicated by the fact that as other agents learn, the environment effectively changes. Moreover, other agents' actions are often not directly observable, and the action taken by the learning agent can strongly bias the range of behaviors that are encountered. Hu and Wellman [69] characterize an agent's belief process in terms of conjectures about the effect of their actions. A conjectural equilibrium is then defined, in which all agents' expectations are realized, and each agent responds optimally to its expectations. They present a multi-agent system where an agent builds a model of the response of others. Their experimental results show that depending on the starting point, the agent may be better or worse off than had it not attempted to learn a model of the other agents.

In [159] the Bazaar negotiation model was presented, which included multi-agent learning through agent interactions. The benefits of learning, if any, on the individual utilities of agents, as well as the overall (joint) system utility were examined. The experimental results suggest that: (a) when all agents learn, the joint system utility is near optimal and agents' individual utilities are very similar; (b) when no agent learns the agents' individual utilities are almost equal but the joint utility is very low (much lower than in the "all agents learn" condition); and (c) when only one agent learns, its individual utility increases at the expense of both the individual utility of the other agents as well as the overall joint utility of the system (i.e., only one agent learning has a harmful overall effect) [159, 160].

3.4. Issues and Future Directions

One of the most important driving forces behind MAS research and development is the Internet: agents are populating the Internet at an increasingly rapid pace. These agents invariably need to interact with one-another in order to meet their designer's objectives.

In such open environments, agents that would like to coordinate with each other (either cooperate or negotiate, for example) face two major challenges: first, they must be able to *find* each other (in an open environment, agents might appear and disappear unpredictably), and once they have done that, they must be able to *inter-operate*.

To address the issue of finding agents in an open environment like the Internet, *middle agents* [88] have been proposed. Each agent *advertises* it capability to some middle agent. A number of different agent types have been identified, including *matchmakers* or *yellow page* agents (that match advertisements to requests for advertised capabilities), *blackboard* agents (that collect requests), and *brokers* (that process both). In preliminary experiments [36], it was seen that the behaviors of each type of middle-agent have certain performance characteristics. For example, while brokered systems are more vulnerable to certain failures, they are also able to cope more quickly with a rapidly fluctuating agent work-force. Middle agents are advantageous since they allow a system to operate robustly in the face of agent appearance and disappearance, and intermittent communications.

To allow agents to inter-operate, a number of *agent communication languages* have been designed [98, 135, 49]. These provide a set of performatives based on speech acts [132]. Though such performatives can characterize message types, efficient languages to express message content that allows agents to "understand" each other have not been effectively demonstrated. Thus the *ontology* problem — that of how can agents share meaning — is still open [62].

Another critical issue is effective allocation of limited resources to multiple agents. For example, we have all experienced large time lags in response to Internet queries because of network congestion. Economics-based mechanisms have been utilized in MAS to address problems of resource allocation (the central theme of economic research) [102, 128, 71]. Economics-based approaches, and market mechanisms in particular, are becoming increasingly attractive to MAS researchers both because of the ready availability of underlying formal models, but also because of their potential applicability in Internet-based commerce. In such approaches, agents are assumed to be self-interested utility maximizers. The areas where economics-based approaches have been applied to MAS research to date are: (a) resource allocation; (b) task allocation; and (c) negotiation. In markets, agents that control scarce resources (labor, raw materials, goods, money) agree to share by exchanging some of their respective resources to achieve some common goal. Resources are exchanged with or without explicit prices. Markets assume that exchange prices are publicly known. In auctions, there is a central auctioneer through which coordination happens. Hence the agents need only exchange minimal amounts of information.

Self interested agents, by definition, simply choose a course of action which maximizes their own utility. In a society of self-interested agents, it is desired that if each agent maximizes it local utility, then the whole society exhibits desirable behavior — in other words, locally good behavior implies globally good behavior. The goal is to design mechanisms for self-interested agents such that if agents follow these mechanisms, the overall system behavior will be acceptable. This is called *mechanism design* [7]. There are, however, many problems facing such a society of self-interested agents. First, agents might overuse and hence congest a shared resource, such as a communications network. This problem is called the *tragedy of the commons* [63]. The problem of the tragedy of commons is usually solved by pricing or taxing schemes. Second, a society of self-interested computational agents can exhibit oscillatory or chaotic behavior [72, 145]. Complex behavior can be exhibited by very simple computational ecosystems. Experimental results indicate that imperfect knowledge suppresses oscillatory behavior at the expense of reducing performance. In addition, enhancing the decision-making abilities of some of the individuals in the system can either improve or severely degrade overall system performance. Moreover, systems can remain in non-optimal meta-stable states for long periods before reaching a globally optimal state. In [145] a similar problem is considered. Two approaches are evaluated, corresponding to heterogeneous preferences and heterogeneous transaction costs. Empirically, the transaction cost case is shown to provide stability with near optimal payoffs under certain conditions. The final problem with self-interested systems is that agents might be untruthful or deceitful in order to increase their individual utility. This may have harmful effect on the whole society. Mechanism design techniques have been reported that make it beneficial for agents to report the truth [105].

4. Applications

Agent technology is rapidly breaking out of universities and research labs, and is beginning to be used to solve real-world problems in a range of industrial and commercial applications. Fielded applications exist today, and new systems are being developed at an increasingly rapid rate. Against this background, the purpose of this section is twofold. First, it aims to identify the main areas where agent-based approaches are currently being used and to provide pointers to some exemplar systems within these areas (section 4.1). Secondly, it aims to anticipate likely future directions of applied agent work and to highlight open issues which need to be addressed if this technology is to fulfill its full potential (section 4.2).

4.1. Key Domains and Exemplar Systems

To date, the main areas in which agent-based applications have been reported are as follows: manufacturing, process control, telecommunication systems, air traffic control, traffic and transportation management, information filtering and gathering, electronic commerce, business process management, entertainment and medical care. Whilst a comprehensive review of all the systems in all of these areas is beyond the scope of this paper (see [22, 83, 114]), we attempt to outline some of the key systems in these areas.

4.1.1. Industrial Applications Industrial applications of agent technology were among the first to be developed, and today, agents are being applied in a wide range of industrial systems:

Manufacturing: Parunak [148] describes the YAMS system (*Y*et *A*nother *M*anufacturing *S*ystem), which applies the Contract Net Protocol (see above) to manufacturing control. The basic problem can be described as follows. A manufacturing enterprise is modelled as a hierarchy of workcells. There will, for example, be workcells for milling, lathing, grinding, painting, and so on. These workcells are further grouped into flexible manufacturing systems (FMS), each of which provides a functionality such as assembly, paint spraying, buffering of products, and so on. A collection of such FMSs is grouped into a factory. A single company or organisation may have many different factories, though these factories may duplicate functionality and capabilities. The goal of YAMS is to efficiently manage the production process at these plants. This process is defined by some constantly changing parameters, such as the products to be manufactured, available resources, time constraints, and so on. In order to achieve this enormously complex task, YAMS adopts a multi-agent approach, where each factory and factory component is represented as an agent. Each agent has a collection of plans, representing its capabilities. The contract net protocol allows tasks (i.e., production orders) to be delegated to individual factories, and from individual factories down to FMSs, and then to individual work cells. Other systems in this area include those for: configuration design of manufacturing products [32], collaborative design [31, 115], scheduling and controlling manufacturing operations [50, 112, 116, 136], controlling a manufacturing robot [113], and determining production sequences for a factory [26, 157].

Process Control: Process control is a natural application for agents, since process controllers are themselves autonomous reactive systems. It is not surprising, therefore, that a number of agent-based process control applications should have been developed. The best known of these is ARCHON, a software platform for building multi-agent systems, and an associated methodology for building applications with this platform [81]. ARCHON has been applied in several process control applications, including electricity transportation management (the application is in use in northern Spain [29]), and particle accelerator control [117]. ARCHON also has the distinction of being one of the world's earliest field-tested multi-agent systems. Other agent-based process control systems have been written for monitoring and diagnosing faults in nuclear power plants [150], spacecraft control [131, 76], climate control [27] and steel coil processing control [101].

Telecommunications: Telecommunication systems are large, distributed networks of interconnected components which need to be monitored and managed in real-time. In what is a fiercely competitive market, telecommunication companies and service providers aim to distinguish themselves from their competitors by providing better, quicker or more reliable services. To achieve this differentiation, they are increasingly turning to state-of-the-art software techniques including agent-based approaches. In one such application, [59], negotiating agents are used to tackle the feature interaction problem. Features in a telecommunication system provide added functionality on top of the basic communication (e.g., call forwarding and caller-id). As new features are being added to the phone network at an ever increasing rate, it is becoming correspondingly more difficult to determine which features interact with, and are inconsistent with, which other features. Therefore, the traditional approach of analysing services at design time and hard-wiring in solutions for all possible interaction permutations is doomed to failure. Given this situation, Griffeth and Velthuijsen [59] decided to adopt a different strategy and tackle the problem on an as-needed basis at run-time. They did this by employing negotiating agents to represent the different entities who are interested in the set up of a call. When conflicts are detected, the agents negotiate with one another to resolve them so that an acceptable call configuration is established. Other problems for which agent-based systems have been constructed include: network control [129, 152], transmission and switching [110], service management [19] and network management [1, 41, 53, 119]. See [153] for a comprehensive review of this area.

Air Traffic Control: Ljunberg and Lucas [95] describe a sophisticated agent-realised air traffic control system known as OASIS. In this system, which is undergoing field trials at Sydney airport in Australia, agents are used to represent both aircraft and the various air-traffic control systems in operation. The agent metaphor thus provides a useful and natural way of modelling real-world autonomous components. As an aircraft enters Sydney airspace, an agent is allocated for it, and the agent is instantiated with the information and goals corresponding to the real-world aircraft. For example, an aircraft might have a goal to land on a certain runway at a certain time. Air traffic control agents are responsible for managing the system. OASIS is implemented using a belief-desire-intention system called DMARS [56].

Transportation Systems: The domain of traffic and transportation management is well suited to an agent-based approach because of its geographically distributed nature. For example, [18] describe a multi-agent system for implementing a future car pooling application. Here there are two types of agent: one representing the customers who require or who can offer transportation, and one representing the stations where customers congregate in order to be picked up. Customer agents inform relevant stations of their requirements (when and where they want to go) and the station agent determines whether their requests can be accommodated and, if so, which car they should be booked into. Other applications in this area are described in [46].

4.1.2. Commercial Applications While industrial applications tend to be highly-complex, bespoke systems which operate in comparatively small niche areas, commercial applications, especially those concerned with information management, tend to be oriented much more towards the mass market.

Information Management: As the richness and diversity of information available to us in our everyday lives has grown, so the need to manage this information has grown. The lack of effective information management tools has given rise to what is colloquially known as the *information overload* problem. Put simply, the sheer volume of information available to us via the Internet and World-Wide Web (WWW) represents a very real problem. The potential of this resource is enormous, but the reality is often disappointing. There are many reasons for this. Both human factors (such as users getting bored or distracted) and organisational factors (such as poorly organised pages with no semantic mark-up) conspire against users attempting to use the resource in a systematic way. We can characterise the information overload problem in two ways:

- *Information filtering*: Every day, we are presented with enormous amounts of information (via email and usenet news, for example), only a tiny proportion of which is relevant or important. We need to be able to sort the wheat from the chaff, and focus on the information we need.

- *Information gathering*: The volume of information available prevents us from actually finding information to answer specific queries. We need to be able to obtain information that meets our requirements, even if this information can only be collected from a number of different sites.

One important contributing factor to information overload is almost certainly that an end user is required to constantly direct the management process. But there is in principle no reason why such searches should not be carried out by agents, acting autonomously to search the web on behalf of some user. The idea is so compelling that many projects are directed at doing exactly this — indeed this is probably the single most active area for agent applications. Two typical projects are:

- Maxims is an electronic mail filtering agent which "learns to prioritise, delete, forward, sort, and archive mail messages on behalf of a user" [97, p35]. It works by "looking over the shoulder" of a user as he or she works with their email reading program, and uses every action the user performs as a lesson. Maxims constantly makes internal predictions about what a user will do with a message. If these predictions turn out to be inaccurate, then Maxims keeps them to itself. But when it finds it is having a useful degree of success in its predictions, it starts to make suggestions to the user about what to do.
- The WARREN financial portfolio management system, is a multi-agent system that integrates information finding and filtering in the context of supporting a user manage her financial portfolio. The system consists of agents that cooperatively self-organize to monitor and track stock quotes, financial news, financial analysts reports, and company earnings reports in order to appraise the portfolio owner of the evolving financial picture. The agents not only answer relevant queries but also continuously monitor available information resources for the occurrence of interesting events (e.g., a particular stock has gone up past a threshold) and alert the portfolio manager agent or the user.

Other applications in this area include: WEBMATE [25], a personal assistant that learns user interests and on the basis of these compiles a personal newspaper, a personal assistant agent for automating various user tasks on a computer desktop [20], a home page finder agent [42], a web browsing assistant [94] and an expert locator agent [85].

Electronic Commerce: Currently, commerce is almost entirely driven by human interactions; humans decide when to buy goods, how much they are willing to pay, and so on. But in principle, there is no reason why some commerce cannot be automated. By this, we mean that some commercial decision making can be placed in the hands of agents. Although widespread electronic commerce is likely to lie some distance in the future, an increasing amount of trade is being undertaken by agents. As an example, [24] describes a simple "electronic marketplace" called Kasbah. This system realises the marketplace by creating "buying" and "selling" agents for each good to be purchased or sold respectively. Commercial transactions then take place by the interactions of these agents. Other commerce applications include BargainFinder [87] an agent which discovers the cheapest CDs, Jango [37] a personal shopping assistant able to search on-line stores for product availability and price information, MAGMA [147] a virtual marketplace for electronic commerce, and several agent-based interactive catalogues [130, 142].

Business Process Management: Company managers make informed decisions based on a combination of judgement and information from many departments. Ideally, all relevant information should be brought together before judgement is exercised. However obtaining pertinent, consistent and up-to-date information across a large company is a complex and time consuming process. For this reason, organisations have sought to develop a number of IT systems to assist with various aspects of the management of their business processes. Project ADEPT [82] tackles this problem by viewing a business process as a community of negotiating, service providing agents. Each agent represents a distinct role or department in the enterprise and is capable of providing one or more services. Agents who require a service from another agent enter into a negotiation for that service to obtain a mutually acceptable price, time, and degree of quality. Successful negotiations result in binding agreements between agents. Other application in this area include a system for supply chain management [51], a system for managing heterogeneous workflows [75] and a system of mobile agents for inter-organisational workflow management [100].

4.1.3. Entertainment Applications The leisure industry is often not taken seriously by the computer science community. Leisure applications are frequently seen as somehow peripheral to the "serious" applications of computers. And yet leisure applications such as computer games can be extremely challenging and lucrative. Agents have an obvious role in computer games, interactive theatre, and related virtual reality applications: such systems tend to be full of semi-autonomous animated characters, which can naturally be implemented as agents.

Games: Grand and Cliff [58] built the highly successful Creatures game using agent techniques. Creatures provides a rich, simulated environment containing a number of synthetic agents that a user can interact with in real-time. The agents are intended to be sophisticated pets whose development is shaped by their experiences during their lifetime. Wavish et al., [151] also describe several applications of agent technology to computer games.

Interactive Theatre and Cinema: By interactive theatre and cinema, we mean a system that allows a user to play out a role analogous to those played by real, human actors in plays or films, interacting with artificial, computer characters that have the behavioural characteristics of real people. Agents that play the part of humans in theatre-style applications are often known as *believable* agents – software programs "that provide the illusion of life, thus permitting [an] audience's suspension of disbelief" [6]. A number of projects have been set up to investigate the development of such agents [146, 65, 92, 48]).

4.1.4. Medical Applications Medical informatics is a major growth area in computer science: new applications are being found for computers every day in the health industry. It is not surprising, therefore, that agents should be applied in this domain. Two of the earliest applications are in the areas of patient monitoring and health care.

Patient Monitoring: The GUARDIAN system [67] is intended to help manage patient care in the Surgical Intensive Care Unit (SICU). The system was motivated by two concerns: first, that the patient care model in a SICU is essentially that of a team, where a collection of experts with distinct areas of expertise cooperate to organise patient health care; and second, that one of the most important factors in good SICU patient health care is the adequate sharing of information between members of the critical care team. In particular, specialists tend to have very little opportunity to monitor the minute-by-minute status of a patient; this task tends to fall to nurses, who, in contrast, often do not have the expertise to interpret the information they obtain in the way that an appropriate expert would. The GUARDIAN system distributes the SICU patient monitoring function among a number of agents, of three different types: *perception/action* agents — responsible for the interface between GUARDIAN and the world, mapping raw sensor input into a usable symbolic form, and translating action requests from Guardian into raw effector control commands; *reasoning* agents - responsible for organising the system's decision making process; and *control* agents — of which there will only ever be one, with overall, top-level control of the system. These agents are organised into hierarchies, and the system as a whole is closely based on the blackboard model of control.

Health Care: A prototypical agent-based distributed medical care system is described in [70]. This system is designed to integrate the patient management process, which typically involves many individuals. For example, a general practitioner may suspect that a patient has breast cancer, but this suspicion cannot be confirmed or rejected without the assistance of a hospital specialist. If the specialist confirms the hypothesis, then a care programme must be devised for treating the patient, involving the resources of other individuals. The system allows a natural representation of this process, with agents mapped onto the individuals and, potentially, organisations involved in the patient care process.

4.2. Future Directions

The aforementioned systems can be considered as the first wave of agent-based applications. In addition to providing solutions to meet real-world needs, they demonstrate that agent-based systems are a useful and powerful solution technology[2]. That is, the conception of (multiple) autonomous problem solvers interacting in various ways to achieve individual and system goals is a useful software engineering abstraction (just as objects and abstract data types are). The abstraction is useful to the extent that it enables software engineers to do more or to do things more cheaply.

However, these developments also show that designing and building agent systems is difficult. They have all the problems associated with building traditional distributed, concurrent systems, and have the additional difficulties which arise from having flexible and sophisticated interactions between autonomous problem solving components. For these reasons, most extant agent system applications are built by, or in consultation with, designers and developers who are themselves active in the agent research community. Whilst this may suffice for a niche software technology, we feel agents have the potential to be far more ubiquitous than this. Indeed, we firmly believe that agent technology has the potential to enter the mainstream of software engineering solutions (in the same way that object-oriented technology has). However for this to occur, it must be possible for professional software engineers to design and build multi-agent systems. The big question then becomes one of how this is achieved.

At this time, there are two major technical impediments to the widespread adoption of agent technology: (i) the lack of a systematic methodology enabling designers to clearly specify and structure their applications as multi-agent systems; and (ii) the lack of widely available industrial-strength multi-agent system toolkits. The former means that most extant applications have been designed in a fairly ad hoc manner — either by borrowing a methodology (typically an object-oriented one) and trying to shoe-horn it to the multi-agent context or by working without a methodology and designing the system based on intuition and past experience. Clearly this situation is unsatisfactory. What is required is a systematic means of analysing the problem, of working out how it can be best structured as a multi-agent system, and then determining how the individual agents can be structured. The latter impediment means that most multi-agent system projects expend significant development effort building up basic infrastructure before the main thrust of agent and inter-agent development can commence. Again, this is an unsustainable position. The position can be alleviated to a certain extent by exploiting existing technologies (such as CORBA) as and where appropriate — rather than re-inventing the wheel as often happens at the moment. However, we believe that still greater support is needed for the process of building agent-level features. Thus, a toolkit (or a flexible set of tools) is required, providing facilities for: specifying an agent's problem solving behaviour, specifying how and when agents should interact, and visualising and debugging the problem solving behaviour of the agents and of the entire system.

The other major impediment to the widespread adoption of agent technology has a social as well as a technical aspect. For individuals to be comfortable with the idea of delegating tasks to agents, they must first trust them. Both individuals and organisations will thus need to become more accustomed and confident with the notion of autonomous software components, if they are to become widely used. Users have to gain confidence in the agents that work on their behalf, and this process can take time. During this period, the agent must strike balance between continually seeking guidance (and needlessly distracting the user) and never seeking guidance (and exceeding its authority). Put crudely, an agent must know its limitations.

5. Concluding Remarks

The field of autonomous agents and multi-agent systems is a vibrant and rapidly expanding area of research and development. It represents a melting pot of ideas originating from such areas as distributed computing, object-oriented systems, software engineering, artificial intelligence, economics, sociology, and organisational science. At its core is the concept of autonomous agents interacting with one another for their individual and/or collective good. This basic conceptual framework has become common currency in a range of closely related disciplines, and offers a natural and powerful means of analysing, designing, and implementing a diverse range of software solutions.

Over the past two decades, a number of significant conceptual advances have been made in both the design and implementation of individual autonomous agents, and in the way in which they interact with one another. Moreover, these technologies are now beginning to find their way into commercial products and real-world software solutions. However, despite the obvious potential, there are a number of fundamental research and development issues which remain. As we have indicated, these issues cover the whole gamut of the agents field, and only when robust and scalable solutions have been found will the full potential of agent-based systems be realised.

Notes

1. It should be noted that some recent game theoretic models are directly motivated by considerations of dropping or relaxing some of these assumptions. Although there has been interesting progress reported in the literature (e.g., [84]), the fundamental framework and methodology of game theory remains almost the same and it might be too early to tell whether these new results will reshape the current game theoretic framework.
2. This is true even though most implemented systems use techniques which are towards the simpler end of those reported in the literature.

References

1. M. R. Adler, A. B. Davis, R. Weihmayer, and R. W. Worrest. Conflict resolution strategies for nonhierarchical distributed agents. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence Volume II*, pages 139–162. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.
2. G. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems*. The MIT Press: Cambridge, MA, 1986.
3. G. Agha and C. Hewitt. Concurrent programming using actors. In Y. Yonezawa and M. Tokoro, editors, *Object-Oriented Concurrent Programming*. MIT Press, 1988.
4. G. Agha, P. Wegner, and A. Yonezawa, editors. *Research Directions in Concurrent Object-Oriented Programming*. The MIT Press: Cambridge, MA, 1993.
5. J. F. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning*. Morgan Kaufmann Publishers: San Mateo, CA, 1990.
6. J. Bates. The role of emotion in believable agents. *Communications of the ACM*, 37(7):122–125, July 1994.
7. K. Binmore. *Fun and Games: A Text on Game Theory*. D. C. Heath and Company: Lexington, MA, 1992.
8. R. P. Bonasso, D. Kortenkamp, D. P. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 187–202. Springer-Verlag: Berlin, Germany, 1996.
9. A. H. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA, 1988.
10. G. Booch. *Object-Oriented Analysis and Design (second edition)*. Addison-Wesley: Reading, MA, 1994.
11. J. Bradshaw, editor. *Software Agents*. The MIT Press: Cambridge, MA, 1997.
12. M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press: Cambridge, MA, 1987.
13. M. E. Bratman. Planning and the stability of intentions. *Minds and Machines*, 2:1–16, 1992.
14. M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.
15. R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

16. R. A. Brooks. Intelligence without reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1991.
17. R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.
18. B. Burmeister, A. Haddadi, and G. Matylis. Applications of multi-agent systems in traffic and transportation. *IEE Transactions on Software Engineering*, 144(1):51–60, February 1997.
19. M. Busuoic and D. Griffiths. Cooperating intelligent agents for service management in communications networks. In S. M. Deen, editor, *Proceedings of the 1993 Workshop on Cooperating Knowledge Based Systems (CKBS-93)*, pages 213–226. DAKE Centre, University of Keele, UK, 1994.
20. A. Caglayan, M. Snorrason, J. Mazzu J. Jacoby, R. Jones, and K. Kumar. Open sesame – a learning agent engine. *Applied Artificial Intelligence*, 11(5):393–412, 1997.
21. S. Cammarata, D. McArthur, and R. Steeb. Strategies of cooperation in distributed problem solving. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, Karlsruhe, Federal Republic of Germany, 1983.
22. B. Chaib-draa. Industrial applications of distributed ai. *Communications of the ACM*, 38(11):47–53, 1995.
23. D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.
24. A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, pages 75–90, London, UK, 1996.
25. Liren Chen and Katia Sycara. Webmate : A personal agent for browsing and searching. In *Proceedings of the Second International Conference on Autonomous Agents (Agents 98)*, Minneapolis/St Paul, MN, May 1998.
26. K. T. Chung and C. H. Wu. Dynamic scheduling with intelligent agents. Metra Application Note 105, Metra, Palo Alto, CA, 1997.
27. S. H. Clearwater, R. Costanza, M. Dixon, and B. Schroeder. Saving energy using market-based control. In S. H. Clearwater, editor, *Market Based Control*, pages 253–273. World Scientific: Singapore, 1996.
28. P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.
29. J. M. Corera, I. Laresgoiti, and N. R. Jennings. Using archon, part 2: Electricity transportation management. *IEEE Expert*, 11(6):71–79, 1996.
30. D. D. Corkill and V. R. Lesser. The use of meta-level control for coordination in a distributed problem solving network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 748–756, Karlsruhe, Germany, 1983.
31. M. R. Cutosky, R. E. Fikes R. S. Engelmore, M. R. Genesereth, W. S. Mark T. Gruber, J. M. Tenenbaum, and J. C. Weber. PACT: An experiment in integrating concurrent engineering systems. *IEEE Transactions on Computers*, 26(1):28–37, 1993.
32. T. P. Darr and W. P. Birmingham. An attribute-space representation and algorithm for concurrent engineering. *AI EDAM*, 10(1):21–35, 1996.
33. Davis, R. and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–100, 1983.
34. K. Decker and V. Lesser. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 73–80, San Francisco, CA, June 1995.
35. K. Decker, A. Pannu, K. Sycara, and M. Williamson. Designing behaviors for information agents. In *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, pages 404–412, Marina del Rey, CA, February 1997.
36. K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, 1997.
37. R. Doorenbos, O. Etzioni, and D. Weld. A scaleable comparison-shopping agent for the world wide web. In *Proceedings of the First International Conference on Autonomous Agents (Agents 97)*, pages 39–48, Marina del Rey, CA, 1997.
38. E. H. Durfee. *Coordination of Distributed Problem Solvers*. Kluwer Academic Publishers: Boston, MA, 1988.
39. E. H. Durfee and V. Lesser. Negotiating task decomposition and allocation using partial global planning. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence Volume II*, pages 229–244. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.
40. R. Engelmore and T. Morgan, editors. *Blackboard Systems*. Addison-Wesley: Reading, MA, 1988.
41. B. Esfandiari, G. Deflandre, and J. Quinqueton. An interface agent for network supervision. In *Proceedings of the ECAI-96 Workshop on Intelligent Agents for Telecom Applications*, Budapest, Hungary, 1996.

42. O. Etzioni. Moving up the information food chain: Deploying softbots on the world-wide web. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, OR, 1996.
43. I. A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, Clare Hall, University of Cambridge, UK, November 1992. (Also available as Technical Report No. 273, University of Cambridge Computer Laboratory).
44. I. A. Ferguson. Integrated control and coordinated behaviour: A case for agent models. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 203–218. Springer-Verlag: Berlin, Germany, January 1995.
45. R. E. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2):189–208, 1971.
46. K. Fischer, J. P. Müller, and M. Pischel. Cooperative transportation scheduling: An application domain for dai. *Applied Artificial Intelligence*, 10(1):1–34, 1996.
47. M. Fisher. Representing and executing agent-based systems. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 307–323. Springer-Verlag: Berlin, Germany, January 1995.
48. L. N. Foner. Entertaining agents: A sociological case study. In *Proceedings of the First International Conference on Autonomous Agents (Agents 97)*, pages 122–129, Marina del Rey, CA, 1997.
49. The Foundation for Intelligent Physical Agents. See `http://drogo.cselt.stet.it/fipa/`.
50. K. Fordyce and G. G. Sullivan. Logistics management system: Integrating decision technologies for dispatch scheduling in semi-conductor manufacturing. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, pages 473–516. Morgan Kaufmann Publishers: San Mateo, CA, 1994.
51. M. S. Fox, J. F. Chionglo, and M. Barbuceanu. The integrated supply chain management system. Technical report, Department of Industrial Engineering, University of Toronto, 1993.
52. S. Franklin and A. Graesser. Is it an agent, or just a program? In J. P. Müller, M. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III (LNAI Volume 1193)*, pages 21–36. Springer-Verlag: Berlin, Germany, 1997.
53. F. J. Garijo and D. Hoffmann. A multi-agent architecture for operation and maintenance of telecommunications networks. In *Proceedings of the Twelfth International Conference on AI, Expert Systems and Natural Language*, pages 427–436, Avignon, France, 1992.
54. L. Gasser. Social conceptions of knowledge and action. *Artificial Intelligence*, 43(1), 1991.
55. M. P. Georgeff. Communication and interaction in multi-agent planning. In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*, Washington, D.C., 1983.
56. M. P. Georgeff. Distributed multi-agent reasoning systems (dmars). Technical report, Australian AI Institute, Level 6, 171 La Trobe Street, Melbourne, Australia, 1994.
57. M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA, 1987.
58. S. Grand and D. Cliff. Creatures: Entertainment software agents with artificial life. *Autonomous Agents and Multi-Agent Systems*, 1(2), 1998.
59. N. D. Griffeth and H. Velthuijsen. The negotiating agents approach to run-time feature interaction resolution. In L. G. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunications Systems*, pages 217–235. IOS Press, 1994.
60. Barbara Grosz and Sarit Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86(2):269–3571, 1996.
61. Barbara Grosz and Candace Sidner. Plans for discourse. In Phil Cohen, Jerry Morgan, and Martha Pollack, editors, *Intentions in Communication*, pages 417–444. MIT Press, Cambdrige, MA, 1990.
62. T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.
63. G. Hardin. The tragedy of commons. *Science*, 162:1243, 1968.
64. B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26, 1985.
65. B. Hayes-Roth. Agents on stage: Advancing the state of the art in AI. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 967–971, Montréal, Québec, Canada, August 1995.
66. B. Hayes-Roth and L. Brownston. Multiagent collaboration in directed improvisation. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 148–154, San Francisco, CA, June 1995.
67. B. Hayes-Roth, M. Hewett, R. Washington, R. Hewett, and A. Seiver. Distributing intelligence within an individual. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence Volume II*, pages 385–412. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.

68. C. Hewitt and J. Inman. Dai betwist and between: From intelligent agents to open systems science. *IEEE Trans. SMC*, 21(6):1409–1418, 1991.
69. Junling Hu and Michael P. Wellman. Self-fulfilling bias in multiagent learning. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, Kyoto, Japan, 1996.
70. J. Huang, N. R. Jennings, and J. Fox. An agent-based approach to health care management. *Applied Artificial Intelligence*, 9(4):401–420, 1995.
71. B. A. Huberman and S. H. Clearwater. A multiagent system for controlling building environments. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 171–176, San Francisco, CA, June 1995.
72. B. A. Huberman and T. Hogg. The behavior of computational ecologies. In B. A. Huberman, editor, *The Ecology of Computation*. North-Holland, Amsterdam, 1988.
73. M. Huhns, U. Mukhopadhyay, and L. M. Stephens. DAI for document retrieval: The MINDS project. In M. Huhns, editor, *Distributed Artificial Intelligence*, pages 249–284. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1987.
74. M. Huhns and M. P. Singh, editors. *Readings in Agents*. Morgan Kaufmann Publishers: San Mateo, CA, 1998.
75. M. N. Huhns and M. P. Singh. Managing heterogeneous transaction workflows with cooperating agents. In N. R. Jennings and M. Wooldridge, editors, *Agent Technology: Foundations, Applications and Markets*. Springer-Verlag: Berlin, Germany, 1998.
76. F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6), 1992.
77. N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.
78. N. R. Jennings. Specification and implementation of a belief desire joint-intention architecture for collaborative problem solving. *Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318, 1993.
79. N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 74(2), 1995.
80. N. R. Jennings and J. R. Campos. Towards a social-level characterisation of socially responsible agents. *IEE Transactions on Software Engineering*, 144(1):11–25, February 1997.
81. N. R. Jennings, J. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriolat, P. Skarek, and L. Z. Varga. Using ARCHON to develop real-world DAI applications for electricity transportation management and particle acceleration control. *IEEE Expert*, 11(6):60–88, December 1996.
82. N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien, and M. E. Wiegand. Agent-based business process management. *International Journal of Cooperative Information Systems*, 5(2-3):105–130, 1996.
83. N. R. Jennings and M. Wooldridge. Applying agent technology. In N. R. Jennings and M. Wooldridge, editors, *Agent Technology: Foundations, Applications, and Markets*. Springer-Verlag: Berlin, Germany, 1998.
84. J. S. Jordan. The exponential covergence of bayesian learning in normal form games. *Games and Economic Behavior*, 4:202–217, 1992.
85. H. Kautz, B. Selman, and M. Shah. The hidden web. *AI Magazine*, 18(2):27–35, 1997.
86. S. Kraus, J. Wilkenfeld, and G. Zlotkin. Multiagent negotiation under time constraints. *Artificial Intelligence*, 75(2):297–345, 1995.
87. B. Krulwich. The BargainFinder agent: Comparison price shopping on the internet. In J. Williams, editor, *Bots, and other Internet Beasties*, pages 257–263. Macmillan Computer Publishing: Indianapolis, 1996.
88. D. R. Kuokka and L. P. Harada. Issues and extensions for information matchmaking protocols. *International Journal of Cooperative Information Systems*, 5(2-3):251–274, 1996.
89. Y. Lésperance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl. Foundations of a logical approach to agent programming. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 331–346. Springer-Verlag: Berlin, Germany, 1996.
90. V. R. Lesser, E. H. Durfee, and D. D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transaction on Knowledge and Data Engineering*, 1(1):63–83, 1989.
91. V.R. Lesser. A retrospective view of fa/c distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Artificial Intelligence*, 21(6):1347–1362, December 1991.

92. J. C. Lester and B. A. Stone. Increasing believability in animated pedagogical agents. In *Proceedings of the First International Conference on Autonomous Agents (Agents 97)*, pages 16–21, Marina del Rey, CA, 1997.
93. H. J. Levesque, P. R. Cohen, and J. H. T. Nunes. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 94–99, Boston, MA, 1990.
94. H. Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 924–929, Montréal, Québec, Canada, August 1995.
95. M. Ljunberg and A. Lucas. The OASIS air traffic management system. In *Proceedings of the Second Pacific Rim International Conference on AI (PRICAI-92)*, Seoul, Korea, 1992.
96. Ovum Ltd. Intelligent agents: The next revolution in software, 1994.
97. P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, July 1994.
98. J. Mayfield, Y. Labrou, and T. Finin. Evaluating KQML as an agent communication language. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 347–360. Springer-Verlag: Berlin, Germany, 1996.
99. F. G. M^{c}Cabe and K. L. Clark. `April` — agent process interaction language. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 324–340. Springer-Verlag: Berlin, Germany, January 1995.
100. M. Merz, B. Lieberman, and W. Lamersdorf. Using mobile agents to support inter-organizational workflow management. *Applied Artificial Intelligence*, 11(6):551–572, 1997.
101. K. Mori, H. Torikoshi, K. Nakai, and T. Masuda. Computer control system for iron and steel plants. *Hitachi Review*, 37(4):251–258, 1988.
102. T. Mullen and M. P. Wellman. Some issues in the design of market-oriented agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI 1037)*, pages 283–298. Springer-Verlag: Heidelberg, Germany, 1996.
103. J. P. Müller. *The Design of Intelligent Agents (LNAI Volume 1177)*. Springer-Verlag: Berlin, Germany, 1997.
104. J. P. Müller and M. Pischel. Modelling interacting agents in dynamic environments. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pages 709–713, Amsterdam, The Netherlands, 1994.
105. Roger B. Myerson. Credible negotiation statements and coherent plans. *Journal of economic theory*, 48:264–303, 1989.
106. N. Negroponte. *Being Digital*. Hodder and Stoughton, 1995.
107. A. Newell. The knowledge level. *Artificial Intelligence*, 18(1):82–127, 1982.
108. A. Newell and H. A. Simon. GPS: A program that simulates human thought. In *Lernende Automaten*. R. Oldenbourg, KG, 1961.
109. Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
110. Y. Nishibe, K. Kuwabara, T. Suda, and T. Ishida. Distributed channel allocation in atm networks. In *Proceedings of the IEEE Globecom Conference*, pages 12.2.1–12.2.7, Houston, TX., 1993.
111. G. M. P. O'Hare and N. R. Jennings, editors. *Foundations of Distributed Artificial Intelligence*. Wiley-Interscience: New York, 1996.
112. E. Oliveira, J. M. Fonseca, and A. Steiger-Garcao. MACIV: A DAI based resource management system. *Applied Artificial Intelligence*, 11(6):525–550, 1997.
113. L. Overgaard, H. G. Petersen, and J. W. Perram. Reactive motion planning: a multi-agent approach. *Applied Artificial Intelligence*, 10(1):35–52, 1996.
114. H. V. D. Parunak. Industrial and practical applications of dai. In G. Weiß, editor, *Multi-Agent Systems*. The MIT Press: Cambridge, MA, 1998.
115. H. V. D. Parunak, A. D. Baker, and S. J. Clark. The aaria agent architecture: An example of requirements-driven agent-based system design. In *Proceedings of the First International Conference on Autonomous Agents (Agents 97)*, pages 482–483, Marina del Rey, CA, 1997.
116. H. V. D. Parunak, A. Ward, M. Fleischer, and J. Sauter. A marketplace of design agents for distributed concurrent set-based design. In *Proceedings of the Fourth International Conference on Concurrent Engineering: Research and Applications*, 1997.
117. F. Perriolat, P. Skarek, L. Z. Varga, and N. R. Jennings. Using archon: Particle accelerator control. *IEEE Expert*, 11(6):80–86, 1996.

118. M. E. Pollack and M. Ringuette. Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 183–189, Boston, MA, 1990.
119. A. S. Rao and M. P. Georgeff. Intelligent real-time network management. In *Proceedings of the Tenth International Conference on AI, Expert Systems and Natural Language*, Avignon, France, 1990.
120. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers: San Mateo, CA, April 1991.
121. A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 439–449, 1992.
122. C. Rich and C. Sidner. Collagen: When agents collaborate with people. In *Proceedings of the International Conference on Autonomous Agents (Agents 97)*, Marina del Rey, CA, 1997.
123. J. S. Rosenschein. *Rational Interaction: Cooperation Among Intelligent Agents*. PhD thesis, Computer Science Department, Stanford University, Stanford, CA 94305, 1985.
124. J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, Boston, MA, 1994.
125. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.
126. S. Russell and D. Subramanian. Provably bounded-optimal agents. *Journal of AI Research*, 2:575–609, 1995.
127. T. Sandholm and V. Lesser. Issues in automated negotation and electronic commerce: extending the contract net protocol. In *Proc. First Int. Conf. on Multiagent Systems (ICMAS-95)*, June 1995.
128. Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 256–262, Washington, July 1993.
129. R. Schoonderwoerd, O. Holland, and J. Bruten. Ant-like agents for load balancing in telecommunications networks. In *Proceedings of the First International Conference on Autonomous Agents (Agents 97)*, pages 209–216, Marina del Rey, CA, 1997.
130. R. Schrooten and W. van de Velde. Software agent foundation for dynamic interactive electronic catalogues. *Applied Artificial Intelligence*, 11(5):459–482, 1997.
131. U. M. Schwuttke and A. G. Quan. Enhancing performance of cooperating agents in real-time diagnostic systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 332–337, Chambéry, France, 1993.
132. J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press: Cambridge, England, 1969.
133. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.
134. E. H. Shortliffe. *Computer-Based Medical Consultations*. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1976.
135. I. Smith and P. Cohen. Towards semantics for an agent communication language based on speech acts. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, OR, 1996.
136. F. Sprumont and J. P. Müller. Amacoia: A multi-agent system for designing flexible assembly lines. *Applied Artificial Intelligence*, 11(6):573–590, 1997.
137. L. Steels. Cooperation between distributed agents through self organization. In Y. Demazeau and J.-P. Müller, editors, *Decentralized AI — Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89)*, pages 175–196. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.
138. P. Stone and M. Veloso. Multiagent systems: A survey from the machine learning perspective. *IEEE Transactions on Knowledge and Data Engineering*, Forthcoming, 1998.
139. K. Sycara. Negotiation planning: An AI approach. *European Journal of Operational Research*, 46:216–234, 1990.
140. K. P. Sycara. Resolving goal conflicts via negotiation. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, St. Paul, MN, 1988.
141. K. P. Sycara. Persuasive argumentation in negotiation. *Theory and Decision*, 28:203–242, 1990.
142. K. Takahashi, Y. Nishibe, I. Morihara, and F. Hattori. Intelligent pages: Collecting shop and service information with software agents. *Applied Artificial Intelligence*, 11(6):489–500, 1997.

143. M. Tambe. Recursive agent and agent group tracking in a real-time, dynamic environment. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 368–375, San Francisco, June 1995. AAAI Press.
144. Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.
145. J. Thomas and K. Sycara. Stability and heterogeneity in multi agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, Paris, France, July, 1998.
146. R. Trappl and P. Petta. *Creating Personalities for Synthetic Actors*. Springer-Verlag: Berlin, Germany, 1997.
147. M. Tsvetovatyy, M. Gini, B. Mobasher, and Z. Wieckowski. MAGMA: An agent-based virtual marketplace for electronic commerce. *Applied Artificial Intelligence*, 11(6):501–524, 1997.
148. H. Van Dyke Parunak. Manufacturing experience with the contract net. In M. Huhns, editor, *Distributed Artificial Intelligence*, pages 285–310. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1987.
149. Manuela Veloso, Peter Stone, Kwun Han, and Sorin Achim. Cmunited: A team of robotic soccer agents collaborating in an adversarial environment. In *Proceedings of the The First International Workshop on RoboCup*, Nagoya, Japan, August 1997. JCAI-97.
150. H. Wang and C. Wang. Intelligent agents in the nuclear industry. *IEEE Computer*, 30(11):28–34, 1997.
151. P. Wavish and M. Graham. A situated action approach to implementing characters in computer games. *Applied Artificial Intelligence*, 10(1):53–74, 1996.
152. R. Weihmayer, I. Ghaznavi, and P. Sheridan. A distributed architecture for cooperative management of strategic communications networks. In *Proceedings of the IEE MILCOM'93 Conference*, Boston, MA, 1993.
153. R. Weihmayer and H. Velthuijsen. Intelligent agents in telecommunications. In N. R. Jennings and M. J. Wooldridge, editors, *Agent Technology: Foundations, Applications and Markets*. Springer-Verlag: Berlin, Germany, 1998.
154. G. Weiß, editor. *Multi-Agent Systems*. The MIT Press: Cambridge, MA, 1998.
155. D. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers: San Mateo, CA, 1988.
156. M. Wooldridge. Agent-based software engineering. *IEE Transactions on Software Engineering*, 144(1):26–37, February 1997.
157. M. Wooldridge, S. Bussmann, and M. Klosterberg. Production sequencing as negotiation. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, pages 709–726, London, UK, April 1996.
158. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
159. Dajun Zeng and Katia Sycara. Benefits of learning in negotiation. In *Proceedings of AAAI-97*, Providence, Rhode Island, U.S.A., 1997.
160. Dajun Zeng and Katia Sycara. Bayesian learning in negotiation. *International Journal of Human-Computer Studies*, 48, in press 1998.