The Knowledge Engineering Review, Vol. 13:4, 1998, 353–380
Printed in the United Kingdom

Architectures and applications of intelligent agents: A survey

JÖRG P. MÜLLER

John Wiley and Sons Ltd., International House, Ealing, London W5 5DB, UK. Email: joerg@elec.qmw.ac.uk

Abstract

The objective of this paper is twofold. In its first part, we survey the state of the art in research on *agent architectures*. The architecture of an agent describes its modules and capabilities, and how these operate together. We structure the field by investigating three important research threads, i.e. architectures for reactive agents, deliberative agents and interacting agents. Then we describe various hybrid approaches that reconcile these three threads, aiming at a combination of different features like reactivity, deliberation and the ability to interact with other agents. These approaches are contrasted with architectural issues of recent agent-based work, including software agents, softbots, believable agents, as well as commercial agent-based systems. The second part of the paper addresses software engineers and system designers who are interested in applying agent technology to their problem domains. The objective of this part is to assist these readers in deciding which agent architecture to choose for a specific *application*. We characterise the most important domains to which the different approaches described in the first part have been applied, propose an application-related taxonomy of agents, and give a set of guidelines to select the right agent (architecture) for a given application.

1 Introduction

The term *autonomous agent* appears to be a magic word in the computing world of the 1990s. The concept of autonomous software programs that are capable of flexible action in complex and changing multiagent environments, has refocused the way artificial intelligence as a whole defines itself (Russell & Norvig, 1995), and is about to find its way into industrial software engineering practice. Agent technology is used to model complex, dynamic, and open applications, e.g. in production planning, traffic control, workflow management, and increasingly, the Internet.

At the heart of an autonomous agent is its control architecture, i.e. the description of its module and of how they work together. Over the past few years, numerous architectures have been proposed in the literature, addressing different key features an agent should have. In the first part of this paper, we survey the state of the art in the design of control architectures for autonomous agents. We start with an investigation of architectural issues raised by three influential threads of agent research, i.e., *reactive agents, deliberative agents* and *interacting agents*. Reactive agents, discussed in section 2, are built according to the behaviour-based paradigm, have no – or at most a very simple – internal representation of the world, and provide a tight coupling of perception and action. Deliberative agents (see section 3) are agents in the symbolic artificial intelligence tradition (Newell & Simon, 1976) that have a symbolic representation of the world in terms of categories such as beliefs, goals or intentions, and that possess logical inference mechanisms to make decisions based on their world model. Finally, *interacting agents* are able to coordinate their activities with those of other agents through communication and, in particular, negotiation. Interacting agents have been mainly investigated in distributed AI; they may have explicit representations of other

agents and may be able to reason about them. We give an overview of the developments in interacting agents in section 4.

Each of these threads focuses on one important property of an agent:

- *Reactive agents*: reactivity and real-time behaviour;
- *Deliberative agents*: the ability to act in a goal-directed manner (*proactiveness*, see Wooldridge & Jennings (1995));
- *Interacting agents*: the ability of cooperative social behaviour.

Thus, it is not surprising that research on agent architectures in the 1990s mainly tried to reconcile these properties in layered (mostly hybrid[1]) agent architectures. In section 5, some important layered architectures are overviewed.

Up to the early 1990s, the driving force behind the development of agent architectures was robotics, and many of the architectures described in sections 2–5 were actually developed and evaluated by using robotics applications. Over the past few years, the term *agent* has been increasingly used in different contexts, e.g.

- agents that are designed to be *believable* to humans,
- *software agents* that act on behalf of or assist humans in a great variety of tasks,
- *softbots* that move through the Internet performing tasks there similar to the way robots act in a physical environment.

In section 6, some architectural issues raised by these new developments shall be investigated.

In the second part of this paper, we address the question as to what agent architectures are most suitable for building different types of agent applications. While no complete and undebatable answer to this question can be given so far, it is possible to provide some guidelines that help the system designer select the right (i.e. most appropriate) architecture for a given problem domain (see section 7). Thus, while the first part of this paper addresses a broad audience, the second part is intended for system designers and software engineers interested in agent technology as a software engineering paradigm.

As stated above, the usage of the term *agent* is used to denote a wide variety of concepts, and across the borders of scientific communities. Rather than trying to be complete, in this survey we select and explain a few representative instances of the most important agent-related research directions. Additional references for further reading are provided in each section. For a general introduction to the area, the reader may refer, e.g. to Maes (1994b), Wooldridge and Jennings (1995), Wooldridge et al. (1996), Johnson (1997), Müller et al. (1997) and Huhns and Singh (1998). Franklin and Graesser (1997) overview different agent definitions and suggest a taxonomy; Nwana's (1996) paper is a survey of software agents, also including a taxonomy for this class of agents. Finally, for a more thorough discussion of the historical development of autonomous agents research, the reader is referred to Müller (1996b, Ch. 2), which also presents agent research in the context of important parental disciplines, such as control theory, distributed systems and cognitive psychology.

2 Architectures for reactive agents

In the mid-1980s, a new school of thought emerged that was strongly influenced by behaviourist psychology. Guided by researchers such as Brooks, Chapman and Agre, Kaelbling, and Maes, architectures were developed for agents that were often called *behaviour-based*, *situated* or *reactive*. These agents make their decisions at run-time, usually based on a very limited amount of information, and simple situation-action rules. Some researchers, most notably Brooks with his *subsumption architecture*, denied the need of any symbolic representation of the world; instead,

[1]We call an architecture *hybrid* if it makes use of differing means of representation or mechanisms of control in different layers.

reactive agents make decisions directly based on sensory input. The design of reactive architectures is partly guided by Simon's (1981) hypothesis that the complexity of the behaviour of an agent can be a reflection of the complexity of the environment in which the agent is operating, rather than a reflection of the agent's complex internal design. The focus of this class of system is directed towards achieving *robust* behaviour instead of *correct* or *optimal* behaviour. *Artificial life* (Conway, 1976; Langton, 1989; Maes, 1994b) is a research discipline that strongly builds on reactive, behaviour-based agent architectures. For further reading on architectures for reactive agents, see, e.g., Agre and Chapman (1987, 1990), Brooks (1986, 1990, 1991), Suchman (1987), Ferber (1989), Kaelbling and Rosenschein (1990), Maes (1990b) and Balch and Arkin (1995).

2.1 Brooks: subsumption architectures

Brooks' subsumption architecture (Brooks, 1986, 1991) provides an *activity-oriented* decomposition of the system into independent *activity producers* which are working in parallel. Individual modules (layers) extract only these aspects of the world which are of interest to them. Thus, the representation space is cut into a set of subspaces. Between the subspaces, no representational information is passed. The lowest layers of the architecture are used to implement basic behaviours such as to avoid hitting things, or to walk around in an area. Higher layers are used to incorporate facilities such as the ability to pursue goals (e.g., looking for and grasping things while walking around).

Control is based on two general mechanisms, namely *inhibition* and *suppression*. Control is layered in that higher-level layers subsume the roles of the lower level layers when they wish to take control. Layers are able to substitute (suppress) the inputs to and to remove (inhibit) the output from lower layers for finite, pre-programmed time intervals. The ability (bias) of the robot agent to achieve its higher-level goal while still attending to its lower-level goals (e.g., the monitoring of critical situations) crucially depends upon the programming of inter-layer control, making use of the two control mechanisms. Brooks was successful in building robots for room exploration, map building and route planning. However, to our knowledge, so far there are no subsumption-based robots that can do complex tasks requiring means-end reasoning and/or cooperation.

2.2 Steels: behaviour-based robots

Steels' approach to modelling autonomous agents described in Steels (1990), forgoes any planning and instead refers to the principle of *emergent functionality* brought about by processes of self-organisation which plays an important role in system theory (Nicolis & Prigogine, 1977), biology, physics and chemistry (Babloyantz, 1986).

The fundamental observation is that complex behaviour of the system as a whole can emerge by the interaction of simple individuals with simple behaviour. This describes the phenomenon of *swarm intelligence*. Steels provided the following example for self-organisation in a scenario involving autonomous robots swarming out from a mother ship on a remote planet to collect samples: samples occur in clusters, and the robot agents use simple rules to indicate regions where samples are likely to be found: if an agent carries a sample, it drops crumbs, if it carries none and detects crumbs, it picks up the crumbs again. Thus, paths are built leading to regions with high density of samples. On the other hand, agents take into account the information provided by the crumbs by following the highest concentration of crumbs. By a simulation, Steels shows that the performance of the agents can be remarkably improved by this reactive cooperation method.

As for most other reactive approaches, Steels' model suffers from the fuzziness of the underlying terms such as *self-organisation* and *emerging behaviour*. The extent to which his model can be generalised, and its general usefulness as a model for intelligent agents that are able to deal with a broader range of tasks and environments, is unclear.

2.3 Arkin: the AuRA architecture

AuRA (Autonomous Robot Architecture) (Arkin, 1990) is an architecture for reactive robot navigation that extends Brooks' approach by incorporating different types of domain knowledge to achieve more flexibility and adaptability. AuRA consists of five components: (i) a perception subsystem which provides perceptual input for other modules; (ii) a cartographic subsystem, i.e., a knowledge base for maintaining both *a priori* and acquired world knowledge; (iii) a planning subsystem which consists of a hierarchical planner and a reactive plan execution subsystem; (iv) a motor subsystem providing the interface to the effectors of the robot; and (v) a homeostatic control subsystem which monitors internal conditions of the robot such as its energy level, and which provides this status information both to the planning subsystem and to the motor subsystem.

The coupling of planner and reacting subsystem is similar to Firby's RAP System (see section 5), Wilkins' Cypress system, as described in Wilkins et al. (1994), and the New Millennium Remote Agent Architecture (NMRA, Pell et al., 1997). However, Arkin's approach sticks to reactivity as its main focus; the underlying representation by a potential field is very application-specific and lacks generality. Moreover, it is hard to see how models of other agents could be incorporated into the architecture apart from treating them as obstacles in a potential field. Thus, as in basically all reactive approaches, the cooperative abilities of AuRA robots do not exceed that of simple grouping or following behaviours (see also Mataric (1993) and Balch & Arkin (1995)). There is no way of expressing goals or even cooperative or synchronised plans.

2.4 Maes: dynamic action selection

Maes (1989, 1990b) presented a model of action selection in dynamic agents, i.e., a model the agent can use to decide what to do next. Driven by the drawbacks of both purely deliberative agents and purely situated agents, Maes argues in favour of introducing the notion of goals for situated agents. However, in contrast to traditional symbolic approaches, her model is based on the idea of describing action selection as an emergent property of a dynamics of activation and inhibition among the different actions the agent can execute. The model eschews any global control arbitrating among the different actions. An agent is described by a set of *competence modules*; these correspond to the notion of operators in classical AI planning. Each module is described by preconditions, expected effects (add and delete lists), and a level of activation. Modules are arranged in a network by different types of links: successor links, predecessor links and conflictor links. A successor link $a \xrightarrow{s} b$ denotes that a provides the precondition for b. A predecessor link between two modules a and b is defined as $a \xrightarrow{p} b$ iff $b \xrightarrow{s} a$. Finally, a conflictor link between two modules a and b, $a \xrightarrow{c} b$ denotes that a disables b by destroying b's precondition.

Modules use these links to activate or inhibit each other in three basic ways: first, the *activation of successors* occurs by an executable module spreading activation forward. This method implements the concept of enforcing sequential actions. Secondly, the *activation of predecessors* provides a simple backtracking mechanism in case of a failure. Thirdly, the *inhibition of conflictors* resolves conflicts among modules. To avoid cyclic inhibition, only the module with the highest activation level is able to inhibit others. Activation messages increases the activation value of a module. If the activation value of a module exceeds the threshold specified by the activation level, and if its preconditions are satisfied, the module will take action.

Maes' approach extends purely reactive approaches by introducing the useful abstraction of goals. However, the underlying process of emergence is not yet fully understood, and the system behaviour resulting from it is difficult to understand, predict and verify.

3 Architectures for deliberative agents

Most agent models in AI are based on Simon and Newell's physical symbol system hypothesis (Newell & Simon, 1976) in their assumption that agents maintain an internal representation of their

world, and that there is an explicit mental state which can be modified by some form of symbolic reasoning. These agents are often called *deliberative agents*. AI planning systems (Fikes et al., 1971; Sacerdoti, 1975; Wilkins, 1988; Ambros-Ingersson & Steel, 1990) can be regarded as the predecessor of today's research on deliberative agents (see Müller (1996b) for an overview). Over the past few years, an interesting research direction has explored the modelling of agents based on Beliefs, Desires and Intentions (BDI architectures). In this section, we give some prominent examples of deliberative agents. Further examples of deliberative agent architectures can be found in Georgeff and Lansky (1986), Laird et al. (1987), Sanborn and Hendler (1990), Cohen et al. (1989), Hayes-Roth (1990, 1995), Allen et al. (1990), Drummond and Kaelbling (1990), McDermott (1991), Wooldridge and Jennings (1995), Wooldridge et al. (1996), Kinny et al. (1996), Gmytrasiewicz (1996), Norman and Long (1996) and Müller et al. (1997).

3.1 Bratman et al.: IRMA

IRMA (Bratman et al., 1987) is an architecture for resource-bounded agents that describes how an agent selects its course of action based on explicit representations of its perception, beliefs, desires and intentions. The architecture incorporates a number of modules, including an intention structure, which is basically a time-ordered set of partial, tree-structured plans, a means-end reasoner, an opportunity analyser, a filtering process and a deliberation procedure. As soon as the agent's beliefs are updated by its perception, the opportunity analyser is able to suggest options for action based on the agent's beliefs. Further options are suggested by the means-end reasoner from the current intentional structure of the agent. All available options run through a filtering process, where they are tested for consistency with the agent's current intentional structure. Finally, options that pass the filtering process successfully are passed to a deliberation process that modifies the intention structure by adopting a new intention, i.e., by committing to a specific partial plan.

The IRMA model embodies two different views on plans: on the one hand, the plans that are stored in the plan library can be looked upon as beliefs the agent has about what actions are useful for achieving its goals. On the other hand, the set of plans the agent has currently adopted define its local intentional structure. This second view of plans as intentions has become now the most accepted paradigm in research on BDI architectures.

IRMA takes a pragmatic stance towards BDI architectures. In particular, it does not provide an explicit formal model for beliefs, goals, and intentions nor for their processing.[2] Thus, the contribution of IRMA has rather been the definition of a control framework for BDI-style agents which served as a basis to many subsequent formal refinements of BDI-concepts.

3.2 Rao and Georgeff: a formal BDI model

Anand Rao and Michael Georgeff (Rao & Georgeff, 1991b) formalised the BDI model, including the definition of the underlying logics, the description of belief, desire and intentions as modal operators, the definition of a possible worlds semantics for these operators, and an axiomatisation defining the interrelationship and properties of the BDI-operators. In contrast to most philosophical theories, Rao and Georgeff have treated intentions as first-class citizens, i.e., as a concept which has equal status to belief and desire. This allows the representation of different types of rational commitment based on different properties on the persistence of beliefs, goals[3] and intentions.

The world is modelled using a temporal structure with branching time future and linear past, a so-called time tree. Situations are defined as particular time points in particular worlds. Time points are

[2]Bratman (1987) gave a theory of intentions from a philosophical point of view.

[3]A note on the relationship between the two terms *goal* and *desire*: whereas desire is an abstract notion that specifies preference over world states, the goals of an agent are defined as a consistent subset of desires. Additionally, it is often required that an agent believes its goals to be achievable (*realism*). Rao and Georgeff have been somewhat inconsistent over time, recently tending to use the more common term *goal* instead of the technically correct term *desire*.

transformed into one another by *events*. There are both primitive and non-primitive events; the latter are useful to model partial and hierarchical plans that are decomposable into subplans and, finally, into primitive actions. There is a distinction between *choice* and *chance*, i.e, between the ability of an agent to deliberately select its actions from a set of alternatives and the uncertainty of the outcome of actions, where the determination is made by the environment rather than by the agent.

The formal language describing these structures is a variation of the Computation Tree Logic (CTL*) (Emerson & Srinivasan, 1989). There are two types of formulae, namely *state formulae* which are evaluated at specific time points, and *path formulae* which are evaluated over a path in a time tree.

Semantics is defined in three parts: a semantics for state and path formulae, a semantics of events, and a semantics of beliefs, goals, and intentions. It is specified by an interpretation M that maps a standard first-order formula into a domain and into truth values, and a possible-worlds semantics for mental modalities by introducing accessibility relations for beliefs, goals and intentions.

Beliefs are axiomatised in the standard weak-S5 (KD45) model system (see Meyer et al., 1991). The D and K axioms are assumed for goals and intentions. That means that goals and intentions are closed under implication and that they have to be consistent. The original axiomatisation provided by Rao and Georgeff (1991a) suffers from the well-known problem of logical omniscience, since, due to the necessitation rule, an agent must believe all valid formulae (see Vardi (1986)), intends them and has the goal to achieve them.

In Rao and Georgeff (1991b), different *commitment strategies* (i.e., relationships between the current intentions of an agent and its future intentions) were discussed: *blind commitment*, *single-minded commitment* and *open-minded commitment*. In Kinny and Georgeff (1991), an empirical evaluation of these strategies is given.

3.3 Rao and Georgeff: an interpreter for a BDI agent

A major criticism of the BDI theory as presented in Rao and Georgeff (1991a) is that the multi-modal BDI logics do not have complete axiomatisations, and that no efficient implementations are available for them (Rao & Georgeff (1995); hence, so far they had little influence on the actual implementation of BDI-systems. In Rao and Georgeff (1992), the authors address this criticism by providing an abstract interpreter for a BDI agent. An abstract agent interpreter is specified that embodies the essential modules of Bratman's BDI agent (see subsection 3.1). It describes the control of an agent by a processing cycle:

```
BDI Interpreter
initialise-state();
repeat
   options := option-generator(event-queue);
   selected-options := deliberate(options);
   update-intentions(selected-options);
   execute();
   get-new-external-events();
   drop-successful-attitudes();
   drop-impossible-attitudes();
end repeat
```

In each cycle, the event queue is looked up by the interpreter. A set of options is generated, i.e., goals that the agent could potentially pursue given the current state of the environment. The set of options is extended by the options that are generated by the deliberator. Finally, the intention formation step is taken in the procedure *update-intentions*. A subset of the options determined so far is selected as intentions, and the agent commits to the associated course of action. If the agent has committed to perform an executable action, the actual execution is initiated. The cycle ends by incorporating

new events into the event queue, and by checking the current goals (options) and intentions whether they have been achieved, or whether they are impossible (in the case of desires) or unrealisable (in case of intentions).

In more recent work, Rao and Georgeff (1995) have proposed a number of restricting assumptions and representation choices to their model to obtain a *practical architecture*:

- Instead of allowing arbitrary formulae for beliefs and goals, these are restricted to be ground sets of literals with no disjunctions or implications.
- Only beliefs about the current state of the world are explicitly represented, denoting the agent's current beliefs that are prone to change over time.
- Information for means-end reasoning, i.e., about means of reaching certain world states and about the options available to the agent for proceeding towards the achievement of its goals, is represented by plans.
- The intentions of an agent are represented implicitly by the agent's run-time stack.

The *Procedural Reasoning System* (Georgeff & Lansky, 1986, 1989) and dMARS (Kinny et al., 1996) are implementations of a BDI architecture based on these assumptions. Wilkins' Cypress system (see Wilkins et al. (1994)) is based on a hybrid architecture (see also section 5) using PRS in conjunction with the SIPE planner (Wilkins, 1988).

3.4 Shoham and Thomas: agent-oriented programming

Shoham (1993) proposed the framework of *agent-oriented programming* (AOP). He presented a class of agent languages that are based on a model looking upon an agent as "an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments". Thus, Shoham adopts the notion of an *intentional stance* as proposed by Dennett (1987).

An AOP system is defined by three components: (i) a formal language for describing mental states of agents; (ii) a programming language with a semantics corresponding to that of a mental state; and (iii) an *agentifier*, i.e., a mechanism for turning a device into what can be called an agent, and which can thus bridge the gap between low-level machine processes and the intentional level of agent programs. In the research published so far, Shoham focused on the former two elements of AOP.

The formal language comprises the mental categories of beliefs, obligations, and capability. Obligation largely coincides with Rao and Georgeff's notion of intention and commitment. Capability is not directly represented as a mental concept in the BDI architecture; rather, it is covered by plans to achieve certain goals.

The programming of agents is viewed as the specification of conditions for making commitments. The control of an agent is implemented by a generic *agent interpreter* running in a two-phase loop. In each cycle, first the agent reads current messages and updates its mental state; second, it executes its current commitments, possibly resulting in further modifications of its beliefs.

AGENT0 is a simple instance of the generic interpreter. The language underlying AGENT0 comprises representations for facts, unconditional and conditional actions (both of which can be private or communicative), and commitment rules which describe conditions under which the agent will enter into new commitments based on its current mental state and on the messages received by other agents. Messages are structured according to message types; admissible message types in AGENT0 are `INFORM`, `REQUEST` and `UNREQUEST`. The corresponding agent interpreter instantiates the basic loop by providing functions for updating beliefs and commitments. Beliefs are updated or revised as a result of being informed or of executing an action. Commitments are updated as a result of changing beliefs or of `UNREQUEST` messages received by other agents.

AGENT0 is a very simple language which was not meant for building interesting applications. Important aspects of agenthood have been neglected: it does not account for motivation, i.e., for how goals of agents come into being, nor for decision-making, i.e., how the agent selects among alternative options. The PLACA language (Thomas, 1993, 1995) extends AGENT0 by introducing knowledge about goals the agent can achieve, and refines the basic agent cycle by adding a time-

dependent step of plan construction and plan refinement. PLACA adopts Bratman's view of plans as intentions, i.e., the agent has a set of plans; its intentions are described by the subset of plans the agent has committed to. Whereas PLACA clearly extends the expressiveness of AGENT0 by providing the notion of plans, it does not address other restrictions such as motivation, decision-making, and the weak expressiveness of the underlying language.

4 Architectures for interacting agents

Distributed Artificial Intelligence (DAI)[4] deals with coordination and cooperation among distributed intelligent agents. So far, its focus has been on the coordination process itself and on mechanisms for cooperation among autonomous agents rather than on the structure of these agents. However, some recent work deals with the incorporation of cooperative abilities into an agent framework. In the following, four prominent approaches are described in more detail: Fischer's MAGSY, the GRATE* architecture by Jennings, Steiner's MECCA system, and the COSY architecture developed by Sundermeyer and Burmeister. For further reading, see, e.g., Georgeff (1983), Finin and Fritzson (1994), Rosenschein and Zlotkin (1994), McCabe and Clark (1995), Barbuceanu and Fox (1996), Chaib-Draa (1996) and Mayfield et al. (1996).

4.1 Fischer: MAGSY

MAGSY (Fischer, 1993) is a rule-based language for designing agents. The MAGSY agent architecture is fairly simple. A MAGSY agent consists of a set of facts representing its local knowledge, a set of rules representing its strategies and behaviour, and a set of services that define the agent's interface. An agent can request a service offered by another agent by communication.

Fischer demonstrates the applicability of MAGSY by the application of decentralised cooperative planning in an automated manufacturing environment. Agents are, e.g., robots, and different types of machines like heating cells or welding machines. The domain plans of the robots are represented as Petri nets, which are translated into a set of rules, so-called *behaviours*. These behaviours are procedures that interleave planning with execution.

The MAGSY language enables the efficient and convenient implementation of multiagent systems. It provides a variety of useful services and protocols to establish multiagent communication links. Clearly, MAGSY inherits both the positive and the negative properties of rule-based programming languages: on the one hand, there is concurrency and the suitability for modelling reactive agents; on the other, there is the flat knowledge representation and the awkward way to represent sequential programs. Cooperation between agents is hard-wired by connections between the Petri nets representing behaviours of different agents. Thus, MAGSY does not support reasoning about cooperation.

*4.2 Jennings: GRATE**

The focus of Jennings' work on the GRATE* architecture (Jennings, 1992b) was on cooperation among possibly preexisting and independent intelligent systems, through an additional *cooperation knowledge layer*. The problem solving capability of agents were extended by sharing information and tasks among each other. GRATE* is an architecture for the design of interacting problem solvers. A general description of cooperative agent behaviour is represented by built-in knowledge. Domain-dependent information about other agents is stored in specific data structures (*agent models*).

[4]See Bond and Gasser (1988) and Gasser and Huhns (1989) for collections of papers that provide a good overview of DAI research by the end of the 1980s. More recent work on DAI can be found in the annual proceedings of the Distributed AI Workshop (until 1994), and in the proceedings of the International Conference on Multiagent Systems (ICMAS) (since 1995).

GRATE* consists of two layers, a *cooperation and control layer* and a *domain level system*. The latter can be preexisting or purpose built; it provides the necessary domain functionality of the individual problem solver. The former layer is a meta-controller operating on the domain level system in order to ensure that its activities are coordinated with those of others in the multiagent system (Jennings, 1992a). Agent models hold different types of knowledge: the *acquaintance model* includes knowledge the agent has about other agents; the *self model* comprises an abstracted perspective of the local domain level system, i.e., of the agent's skills and capabilities. The cooperation and control layer consists of three submodules, representing the interplay between local and cooperative behaviour: The *control module* is responsible for the planning, execution and monitoring of local tasks. The *cooperation module* handles processes of cooperation and coordination with other agents. The *situation assessment module* forms the interface between local and social control mechanisms. It is thus responsible for the decision to choose local or coordinated methods of problem solving.

Clearly, Jennings' focus was on the cooperation process. However, he went beyond work discussed before by defining a two-layer architecture that embeds cooperation into a domain level system.[5] The architecture does not address more subtle questions of agent behaviour, such as how to reconcile reactivity and deliberation. Rather, these problems are expected to be solved within the domain level system.

4.3 Steiner et al: MECCA

In the MECCA architecture (Steiner et al., 1993; Lux & Steiner, 1995; Lux, 1995), an agent is regarded as having an application-dependent *body*, a *head* whose purpose is to actually agentify the underlying system, and a *communicator* which establishes physical communication links to other agents. This view supports the construction of multiagent systems from second principles. Agent modelling is addressed in the design of the agent's head. It is described by a *basic agent loop* consisting of four parallel processes: *goal activation*, *planning*, *scheduling* and *execution*. In the goal activation process, relevant situations (e.g., user input) are recognised and goals are created that are input to the planning process. There, a partially ordered plan structure is generated corresponding to a set of possible courses of action the agent is allowed to take. The scheduler instantiates (serialises) this partially ordered event structure by assigning time points to actions. The execution of actions is initiated and monitored by the execution process.

All control processes in the basic loop may involve coordination with other agents, leading to joint goals, plans, and commitments, and to the synchronised execution of plans. Cooperation is based on speech act theory: MECCA provides a set of cooperation primitives (e.g., `INFORM`, `PROPOSE`, `ACCEPT`) which are treated by the planner as actions, i.e., whose semantics can be described by preconditions and effects. This allows the planner to reason about communication with other agents as a means of achieving goals (Lux and Steiner, 1995). Moreover, cooperation primitives are the basic building blocks of communication protocols, so-called cooperation methods.

4.4 Sundermeyer et al: COSY

The COSY agent architecture (Burmeister & Sundermeyer, 1992) describes an agent by *behaviours*, *resources* and *intentions*. The behaviour of an agent is classified into perceptual, cognitive, communicative and effectoric, each of which is simulated by a specific component in the COSY architecture. Resources include cognitive resources such as knowledge and belief, communication resources such as low-level protocols and communication hardware, and physical resources, e.g., the gripper of a robot. Intentions are used in a sense that differs from Cohen and Levesque (1990) and

[5]Note that the separation of cooperation knowledge from domain knowledge is a theme that can be found in early DAI work, most prominently in the definition of the contract net (Davis & Smith, 1983).

Rao and Georgeff (1991a): there are strategic intentions modelling an agent's long-term goals, preferences, roles and responsibilities, and tactical intentions that are directly tied to actions, representing an agent's commitment to his chosen course of action.

The individual modules of COSY are ACTUATORS, SENSORS, COMMUNICATION, MOTIVATIONS and COGNITION. The former three are domain-specific modules with their intuitive functionality. The motivations module implements the strategic intentions of an agent. The cognition module evaluates the current situation and selects, executes and monitors actions of the agent in that situation. It consists of four subcomponents, a *Knowledge Base*, *Script Execution Component*, *Protocol Execution Component* and *Reasoning and Deciding Component*. The application specific problem solving knowledge is encoded into plans. There are two types of plans stored in a plan library: *scripts* describing stereotypical courses of action to achieve certain goals, and *cooperation protocols* describing patterns of communication (Burmeister et al., 1993). Scripts are monitored and executed by the Script Execution Component, handing over the execution of primitive behaviours to the actuators, and protocols to the Protocol Execution Component. The Reasoning and Deciding Component is a general control mechanism, monitoring and administering the reasoning and decisions concerning task selection and plan selection, including the reasoning and decisions concerning intra-script and intra-protocol branches. Haddadi (1996) has extended this work by providing a deeper theoretical model. Based on Rao and Georgeff's BDI model, her approach describes a theory of commitments, and defines mechanisms allowing agents to reason about how to exploit potentials for cooperation by communicating with each other.

5 Hybrid agent architectures

The approaches discussed so far suffer from different shortcomings: whereas purely reactive systems have a limited scope insofar as they can hardly implement goal-directed behaviour, most deliberative systems are based on general-purpose reasoning mechanisms which are not tractable, and which are much less reactive. One way to overcome these limitations in practice, which has become popular over the past few years, are layered architectures. Layering is a powerful means for structuring functionalities and control, and thus is a valuable tool for system design supporting several desired properties such as reactivity, deliberation, cooperation and adaptability. The main idea is to structure the functionalities of an agent into two or more hierarchically organised layers that interact with each other to achieve coherent behaviour of the agent as a whole. Layering offers the following advantages:

- It supports modularisation of an agent; different functionalities are clearly separated and linked by well-defined interfaces.
- This makes the design of agents more compact, increases robustness and facilitates debugging.
- Since different layers may run in parallel, the agent's computational capability can be increased in principle by a linear factor.
- Especially, the agent's reactivity can be increased: while planning, a reactive layer can still monitor the world for contingency situations.
- Since different types and partitions of knowledge are required for the implementation of different functionalities, it is often possible to restrict the amount of knowledge an individual layer needs to consider.

These advantages have made layering a popular technique that has been mostly used to reconcile reaction and deliberation. In the following, four layered approaches are presented that extend architectural concepts of separating control and execution inherent in the PRS system (see section 3): architectures based on Firby's RAPs, the planner-reactor architecture proposed by Lyons and Hendriks, Ferguson's Touring Machines architecture, and the INTERRAP architecture developed by the author of this survey. For further reading, we refer to Brooks (1986), Kaelbling (1990), Bürckert and Müller (1991), Firby (1992), Lyons and Hendriks (1992), Dabija (1993), Wilkins et al. (1994), Bonasso et al. (1996), Sloman and Poli (1996) and Davis (1997).

5.1 Firby, Gat, Bonasso: reactive action packages

Firby's work (1989, 1992) has been most influential in research on integrating reaction and deliberation in the area of AI planning and robotics. In this subsection, we outline Firby's original work and two of its recent extensions.

The RAPs System The RAPs (Reactive Action Packages) system describes an integrated architecture for planning and control. The underlying agent architecture consists of three modules, *a planning layer*, the *RAP executor* and a *controller*. The planning layer produces sketchy plans for achieving goals using a world model representation and a plan library. The RAP executor fills in the details of the sketchy plans at run-time. The expansion of vague plan steps into more detailed instructions (methods) at run-time reduces the amount of planning uncertainty, and thus largely simplifies planning. If incorrect methods are selected at run-time, the RAP executor is able to recognise failure[6] and to select alternative methods to achieve the goal. Apart from controlling the process of achieving goals in a reactive manner, and thus providing the interface between subsymbolic continuous and symbolic discrete representation and reasoning, the RAP executor offers a set of abstract *primitive actions* to the planner.

The controller provides two kinds of routines that can be activated by requests from the RAP executor and that deliver results to that module: active sensing routines and behaviour control routines. Sensing routines are useful for providing lacking information about the current world state. Behaviour routines are continuous control processes that change the state of the physical environment. Examples for behaviour routines are collision avoidance, visual tracking, or moving in a specified direction. In a later paper (Firby, 1994), the control of continuous processes (i.e., the interplay of the RAP executor and the controller) is elaborated by describing an extension to the RAPs representation language and the semantics for task nets.

ATLANTIS Gat (1991b, 1992) describes the heterogeneous, asynchronous architecture ATLANTIS that combines a traditional AI planner with a reactive control mechanism for robot navigation applications. ATLANTIS consists of three control components: a *controller*, a *sequencer* and a *deliberator*. The controller is responsible for executing and monitoring the primitive *activities* of the agent. Gat (1991a) defines a language for modelling the often nonlinear and discontinuous control processes. The controller thus connects to the physical sensors and actuators of the system. The deliberator process performs deliberative computations which may be time-consuming, such as planning or world modelling. Between the two components stands the sequencer which initiates and terminates primitive activities by activating and deactivating control processes in the controller, and which maintains the allocation of computational resources to the deliberator, by initiating and terminating deliberation with respect to a specific task. As in the RAPs system, the sequencer maintains a task queue; each task described by a set of methods together with conditions for their applicability. Methods describe either primitive activities or subtasks; in the former case, the corresponding module in the controller is activated; the latter case is handled by recursive expansion. ATLANTIS extends Firby's original work by allowing control of activities instead of primitive actions, and provides a bottom-up flow of control: in RAPS, tasks are installed by the planner whereas they are initiated in the sequencer in Gat's architecture.

The $_3$T architecture Peter Bonasso and colleagues (Bonasso et al., 1996) have defined the layered architecture $_3$T, which enhances the RAPs system by a planner. In particular, $_3$T consists of three control layers: a reactive skill layer, a sequencing layer and a deliberation layer. The reactive skill layer provides a set of situated skills. Skills are capabilities that, if placed in the proper context, achieve or maintain particular states in the world. The sequencing layer is based on the RAPs system. It maintains routine tasks that the agent has to accomplish. The sequencing layer triggers

[6]The underlying assumption is that of a *cognizant failure*: it is not required that no failure occurs, but that virtually all possible failures may be detected if they occur, and that repair methods can be applied to recognised failures.

continuous control processes by activating and deactivating reactive skills. Finally, the deliberation layer provides a deliberative planning capability which selects appropriate RAPs to achieve complex tasks. This selection process may involve reasoning about goals, resources and timing constraints.

Compared to Gat's work, $_3$T uses a more powerful planning mechanism; moreover, reactivity, i.e., the ability to react to time-critical events, is implemented at the skill layer in $_3$T, whereas it is partly a task of the sequencing layer in ATLANTIS. Both of these extensions make the sequencing layer in $_3$T more compact and easier to handle.

5.2 Lyons and Hendriks: planning and reaction

In Lyons and Hendriks (1992), a practical approach towards integrating reaction and deliberation in a robotic domain is introduced based on the planner-reactor model proposed by Drummond and Bresina (1990) in their ERE architecture incorporating planning, scheduling, and control.[7] Whereas Drummond and Bresina's model focused on the *anytime* character of the architecture, Lyons and Hendriks put emphasis on the task of producing timely, relevant actions, i.e., on the task of qualitatively reasonable behaviour.

The basic structure of Lyons and Hendriks' architecture is described by a planner, a reactor and a world (which is, in the spirit of control theory (Dean & Wellman, 1991), looked upon as a part of the system to be described). In contrast to the hybrid approaches discussed so far, in Lyons and Hendriks' model, planning is looked upon as incrementally adapting the reactive system which is running concurrently in a separate process by bringing it into accordance with a set of goals. Thus, the planner can iteratively improve the behaviour of the reactive component.

The reactor itself consists of a network of *reactions*, i.e., sensory processes that are coupled with action processes in a sense that the sensory processes initiate their corresponding action processes in case they meet their trigger conditions. It can act any time independently from the planner, and it acts in real time. The planner can reason about a model of the environment (EM), a description of the reactor (R), and a description of goals (G) that are currently to be achieved by the reactor, as well as constraints imposed by these goals. The task of the planner is to continuously monitor whether the behaviour of R conforms to G. If this is not the case, the planner incrementally changes the configuration of R by specifying *adaptations*. On the other hand, the reactor can send collected sensory data to the planner allowing the latter to predict the future state of the environment. Adaptations of the reactor include removing reactions from the reactor and adding new reactions.

5.3 Ferguson: Touring Machines

Ferguson (1992) describes a layered control architecture for autonomous, mobile agents performing constrained navigation tasks in a dynamic environment. The *Touring Machines* architecture consists of three layers, a *reactive layer*, a *planning layer* and a *modelling layer*. These layers operate concurrently; each of them is connected to the agent's sensory subsystem from which it receives perceptual information, and to the action subsystem to which it sends action commands. The reactive layer is designed to compute hard-wired domain-specific action responses to specific environmental stimuli; thus, it brings about reactive behaviour. On the other hand, the planning layer is responsible for generating and executing plans for the achievement of the longer-term relocation tasks the agent has to perform. Plans are stored as hierarchical partial plans in a plan library; based on a topological map, single-agent linear plans of action are computed by the agent. Planning and execution are interleaved to cope with certain forms of expected failure. The modelling layer provides the agent's capability of modifying plans based on changes in its environment that cannot be dealt with by the replanning mechanisms provided by the planning layer. In addition, the modelling layer provides a framework for modelling the agent's environment, and especially, for building and maintaining mental and causal models of other agents.

[7]A similar paradigm has been proposed by McDermott (1991).

The individual layers are mediated by a control framework that coordinates their access both to sensory input and to action output. This is described by means of a set of context-activated control rules. There are two types of rules: *censors* and *suppressors* (see also Minsky (1986)). Censors filter selected information from the input to the control layers; suppressors filter selected information (i.e., action commands) from the output of the control layers.

The unrestricted concurrent access of the control layers to information and action and the global (i.e., for the agent as a whole) control rules in Ferguson's model imply a high design effort to analyse, predict, and prevent possible interactions among the layer. Since each layer may interact with any other layer in various ways either by being activated through similar patterns of perception, or by triggering contradictory or incompatible actions, a large number of control rules are necessary. Thus, for complex applications, the design of consistent control rules itself is a very hard problem.

5.4 Müller and Pischel: INTERRAP

Like the $_3$T and Touring Machines architectures discussed in the previous sections, INTERRAP (Müller & Pischel, 1994; Müller, 1996b, 1996a) consists of three layers. However, the focus of INTERRAP was to extend the scope of agent control architectures by supporting agent interaction. For this purpose, it offers a separate *cooperative planning layer* on top of the *behaviour-based layer* and the *local planning layer*. The cooperative planning layer implements a cooperation model (see Müller (1996a)). It provides negotiation protocols and negotiation strategies (see also Rosenschein & Zlotkin (1994)). Triggered by control messages from the lower layers, an agent can decide to start a cooperation with other agents by selecting a protocol and a strategy. In Müller (1996b), it has been shown how autonomous robots can solve conflicts by negotiating a joint plan.

As regards the control flow, INTERRAP forgoes global control rules. Rather, it uses two hierarchical control mechanisms. Activity is triggered bottom-up by so-called upward activation requests, whereas execution is triggered top-down (downward commitment posting), i.e., a control layer in INTERRAP will become active only if the layer below it cannot deal with a situation. This competence-based control allows an agent to react adequately to a situation, either by patterns of behaviour, by local planning, or by cooperation. The execution output of the cooperative planning layer are partial plans with built-in synchronisation commands; these are passed to the local planning layer, which outputs calls to procedure patterns of behaviour in the behaviour-based component. The latter component then produces actions.

The strict control in INTERRAP considerably simplifies design; a more flexible architecture, e.g., allowing concurrent activation of the different control layers, would certainly be useful, but the discussion of the approaches in this section shows that this will not come for free, and requires sophisticated mechanisms of coordination and pre-emption.

6 Other approaches

Over the past few years, the notion of an agent has been used in a number of different contexts—often confusing, not only for non-experts. Prominent examples of these are *believable agents* and *software agents* or *softbots*. Also, the term *agent* has become a popular add-on to a number of commercial software products. Looking at these approaches in detail, it appears that they cannot easily be mapped into the categories defined earlier on in this paper, but rather seem orthogonal.

For instance, *believability* is orthogonal to both reactivity and deliberation: what matters for a believable agent is not that it can react especially quickly or in a clever fashion to external events, but rather that it appears to act in a way that makes the human observer believe it has a personality. Therefore, believable agents are considered as a separate category in section 6.1.

The same is true for softbots: like robots, the design of softbots can be based either on a reactive, deliberative or hybrid paradigm. What makes them interesting as a class is the fact that they are

programs rather than physical entities. Therefore, we deal with them as a separate case in section 6.2.

6.1 Believable agents

The Oz project (Bates, 1994; Reilly, 1996) investigates agents that are both autonomous and that can act as believable characters in interactions with humans (e.g., in a computer game). Such agents may not be intelligent and may be not be realistic, but will have a strong personality. The approach to modelling these agents is what is called a *broad but shallow approach* (see also Sloman & Poli (1996) and Davis (1997)): instead of being able to do a small number of things particularly well, the agent is required to cope with a variety of situations occurring in interactions with human users.

The architectural challenges in building believable agents lie in the integration of a large set of shallow capabilities. The Tok architecture (see Bates et al. (1992)) extends the traditional AI approach in various respects: first, it enhances sensing–acting by the notion of body and face changes. Secondly, it maintains an emotional state of the agent with specific emotion types and structures, and with corresponding behavioural features. Emotions can cause changing behaviour of the agent in terms of motivation, action and body state and facial expression. Primary application areas for believable agents are education and entertainment. For instance, believable software agents can help children to learn more effectively (see Hayes-Roth & van Gent (1997)); an example for the use of believable, improvisational agents in entertainment is *Erin the Bartender* described by Extempo Systems Inc. (1997). The pedagogical agent Steve that has been developed in the VET project (Johnson & Rickel (1997)) is another example for research addressing of believable agents, as well as the traditional reconciliation of cognitive and sensorimotor processing.

6.2 Software agents and softbots

The rapid development of the Internet has given rise to new classes of agents: *software agents* (Maes, 1994a; Genesereth & Ketchpel, 1994) and *softbots* (see Etzioni & Weld (1994) and Etzioni (1996)). In the software agents approach, the notion of an agent is used as a metaphor for

> intelligent autonomous system[s] that help[s] the user with certain computer-based tasks [...and that] offer[s] assistance to the user and automate[s] as many of the actions of the user as possible'' (Maes, 1994b).

Thus, software agents and softbots act in a software domain; their task is mostly to assist the user in dealing with information management tasks. Whereas software agents research *à la* Maes puts emphasis on the fact that software act on behalf of a human user, the notion of a softbot is strongly based on the analogy to autonomous robots (Etzioni, 1996). While the architectural requirements of modelling software agents seem indeed very similar to that of traditional (mostly robot-like) agents, there are a number of differences: first, a software environment is in many respects easier to deal with than a hardware environment as it liberates the designer from the problems of a rough physical environment that are related to sensing and physically manipulating the world. Thus, although software agents need to cope with a changing world, the possible ways in which this world may change is somewhat more restricted in practice. Secondly, software agents often interact closely with humans. This increases the importance of dealing with communication and cooperation compared to robotic systems. Both differences seem to justify agent architectures for software agents to focus on the higher level functionalities such as adaptation, cooperation and planning, and to simplify the lower layers of reactivity and physical action.

Current software agent research mostly operates under what Etzioni calls a *useful first* approach, that focuses on useful functionality and defers or abandons the claim of (AI style) intelligence (see Franklin & Graesser (1997) and Petrie (1997) for two interesting recent essays in this context). Etzioni's softbots or Maes' assistant agents doubtlessly still have on their research agenda the long-term goal of combining intelligent with useful behaviour in Internet domains (e.g., by using AI planning techniques in the former case); software agents research at Stanford University (Geneser-

eth & Ketchpel, 1994) focuses on communication between agents, and is strongly based on the typed messages paradigm; the KQML Java agent template is an example of an implementation of a simple agent model based on JAVA (Frost, 1996).

However, many current software agents approaches are using the term *agent* in a very loose and shallow manner: they rely on a definition of an agent as a program that is located in a network and—either through migration or through a service interface—can communicate with other programs on the network. From the architectural point of view, this sort of agents does not seem to be very interesting. The challenge in their development is—as it is the case with *mobile agents* (Straßer et al., 1996)—at a different level.

Recently, there has been a true inflation of the term *agent* being used to feature (semi-) commercial agent-based software products, most of which are JAVA or TELESCRIPT-based, and most of which are based on the mobile code paradigm. These commercial agent products can be divided in two categories: Instances of the first category offer software applications with ''agents inside''. Some of them are adaptive search, profiling and Electronic Commerce tools (e.g., Quarter-Deck WebCompass, Firefly, ShopBot, Zuno VRISKO, Autonomy's search agents) that use the term ''agent'' to denote a program with smart search and matching capabilities. The strength of these approaches lies in their domain functionality; in general, they are not very interesting from the point of view of agent architectures. Products that fall into the second category not only offer solutions for specific application domains, but also try to provide application-independent libraries, languages or tools for modelling agents and agent-based systems (see, e.g., IBM's Aglets (Lange & Chang, 1996), or AgentSoft's LiveAgent (AgentSoft). However, most of these systems so far focus on the macro level, looking upon an agent as consisting of a body that implements the agent's functionality and of a general communication interface (similar to the head-mouth-body architecture described in section 4.3).

There are, however, a few commercial approaches that not only cover the macro level, but also support the modelling of individual agents by providing a theory of agency and a general agent architecture: the best known example is the dMARS system, which offers an agent programming environment based on a BDI model (see section 3), along with a design methodology (Kinny & Georgeff, 1997), and which has been used to model various industrial applications (Rao & Georgeff, 1995). The computational core of dMARS agents is (a further development of) the Procedural Reasoning System (PRS) (Georgeff & Ingrand, 1989). Future releases of dMARS will incorporate a theory of team activity (Kinny et al., 1992), and thus, for the first time, deserve the name of a multiagent development platform. Systems such as dMARS might provide the appropriate trade-off between the sophisticated but not sufficiently useful academic approaches, and today's useful but not very sophisticated software agents.

7 Agent architectures and agent applications

In the first part of this paper, we have provided a survey of architectures for intelligent agents. In this section, we address the needs of system designers who require solutions to application problems and who consider using agent technology for this purpose. Given the wide variety of architectures described so far, we feel that it is important to provide this group of readers with a guide to the right agent architecture for a given problem. Furthermore, to our knowledge, this matter has not been addressed before.

We approach this objective in three steps. In section 7.1, we identify application areas for agent technology starting from the examples presented in the first part of this paper. Then, based on the characteristics of different classes of applications identified in section 7.1, we propose a classification of agents according to different classes of applications (section 7.2). Based on this classification, the third step is to provide some rules of thumb to help a software engineer or system designer decide which agent architecture (or which class thereof) is likely to be appropriate for a certain class of applications (section 7.3). Section 7.4 provides a brief discussion of the agent taxonomy and the guidelines.

7.1 Application areas for intelligent agents

In the absence of any theories that could help us in determining which agent paradigm is most useful for which class of applications, we take an empirical approach: we analyse the main areas of application known from the literature for each of the agent architectures described in the first part of the paper.

The first two columns of Table 1 in Appendix A summarise the main application areas for the architectures under consideration, i.e., for reactive, deliberative, interacting and hybrid architectures, as well as for those approaches summarised as *others*. While we refer to the appendix for details, in this subsection we discuss some observations that can be derived from Table 1.

Observation 1 *Most architectures discussed are used for autonomous control systems*. The first striking observation is that a large percentage of applications are in the area of mobile robots or, more broadly speaking, Autonomous Control Systems (ACS). While this is likely to be explained as a historical coincidence, it is striking to what degree this also affects the more recent hybrid architectures, such as NMRA, $_3$T, INTERRAP or SIM_AGENT. The old AI vision of building humanoid computers is still widespread.

Observation 2 *There is only a limited number of examples of cooperating hardware agents*. The second, and maybe more surprising, observation is that while few researchers will doubt the role of cooperation and agent interaction, our list of applications contains only a few examples that actually use interaction among ACSs as a core ingredient. Where these systems can be found (most notably production planning and flexible transport system applications), in most cases the individual agents have limited autonomy, and the interactions among them are simple (e.g., a decentralised material flow where two machines are fed by a transport robot using material buffers, thus eliminating the nitty-gritty details of real-time interaction). One possible explanation for the small number of applications for cooperating ACSs is that there are still a number of fundamental problems in the modelling of an individual ACS (e.g., at the level of sensorimotor control and the abstraction of input sensor data), that need to be solved before the use of cooperating ACSs in real-world applications becomes practical.[8]

Observation 3 *Distributed resource allocation is a core area for using interacting agents*. The third observation is that a considerable class of applications found in Table 1 deals with distributed resource allocation, routing, and scheduling problems. Examples are logistics and transport planning, production planning systems, as well as workflow management and business process enactment and monitoring systems.

Observation 4 *Cooperative expert systems are a core area for using interacting agents*. A fourth observation is that some of the traditional areas of use of expert systems reappear as application areas of agent technology. Again, this is not very surprising, as a significant part of the momentum behind developing multiagent systems originated from the need to build cooperating expert systems. The most prominent example of this class of applications are diagnosis problems that require systems capable to deal with fuzzy and possibly inconsistent knowledge. The GRATE* architecture described in section 4.2 is an example of an architecture used to solve a class of diagnosis problems in electricity management. Another example is the spacecraft health management component of the NMRA architecture (Pell et al., 1997).

Observation 5 *Mainstream architectures do not sufficiently account for HCI requirements*. A fifth observation is that a small class of applications addresses the design of agents that interact with humans. Given that user modelling has been a well-established discipline in AI over a long time (Kobsa & Wahlster, 1989) and that many of its aspects have been revived by research areas such as

[8]Note that there is a variety of related applications (e.g., multi-arm robots) that have been developed outside the agents community. Also, there is a growing area within AI that tries to solve standard robotics problems by using a large number of small and simple robots instead of one complex robot (Konolige, 1997).

Computer-Supported Cooperative Work (CSCW) and *Human Computer Interaction* (HCI), it is surprising how little effect the requirements of dealing with "human agents" have had on the design of mainstream agent architectures. Those applications in Table 1 that require the ability to deal with human agents are mostly software agent applications, such as entertainment, art, education, personal assistants (meeting scheduling, traffic guidance, secretarial functions) and process monitoring agents. However, there seems to be a potential for robots interacting with or assisting humans (e.g., robotic wheelchair, errand-running, robots "working" with humans in factories or offices).

Observation 6 *Designers of software agents often use other architectures than their hardware colleagues*. The final observation we would like to make in this subsection concerns the wide variety of application areas created by the advent of global computer networks, and, in particular, the World Wide Web. These areas require software agents, such as those described in Section 6, to perform information retrieval, information filtering and resource discovery tasks in a real-world software environment. Requirements that are imposed by these applications are the need for interoperability (legacy systems), user profiling capabilities to provide personalised services, and robustness to cope with ever-changing conditions in the environment (e.g., availability of WWW resources) and with changing or context-dependent user preferences. Looking at Table 1, it is striking that the architectures used to build softbot applications seem to differ largely from those developed to build autonomous robots. This is particularly surprising as the notion of a softbot has been derived from a software program being faced with conditions similar to those a mobile robot is likely to encounter, e.g., uncertainty, huge amounts of information, change. We shall get back to this observation in section 7.3.

7.2 An agent taxonomy

Based on the above observations, in this section we propose a taxonomy of intelligent agents that reflects the different application areas identified above, and that can be used to classify the above agent architectures according to how suitable they are for different application problems.

We suggest a classification of agents according to two dimensions:

- The material state of the agents, i.e.:
  - *Hardware agents*: agents that have a physical *gestalt* and that interact with a physical environment through effectors and sensors. Clearly, hardware agents will use software components.
  - *Software agents*: programs that interact with real or virtual software environments.
- The primary mode of interaction between the agent and its environment:
  - *Autonomous agents*: this perspective of autonomous agents concentrates on two entities and their relationship: the agent itself and its environment. Virtually all autonomous control systems fall into this category.
  - *Multiagents*: the environment of multiagents is classified into two categories, i.e., other agents and non-agents. An agent can use its knowledge about other agents to coordinate its actions with those of others, to make better predictions about the future, or to achieve goals collaboratively.
  - *Assistant agents*: assistant agents primarily interact with (and: act on behalf of) one particular type of other agent, i.e., humans.

The motivation for our choice is as follows: the first dimension (material state) is introduced to comply with Observation 6: if different architectures are used in practice to model hardware agents on the one hand, and software agents on the other, the classification should reflect this.

The second dimension (mode of interaction) aims at complying with the main application areas identified above: Wooldridge et al. (1996) provide a similar separation between (autonomous) agents and multiagents underlying Observations 1 to 4. The fact that existing single-agent

architectures were extended or new architectures were developed to cope with the requirements of multiagent applications in our view makes this distinction useful. In addition, Observation 5 suggests considering agents that primarily interact with humans as an important special case of multiagents.

We refer to section 7.4 for a more detailed discussion of this taxonomy. We end this subsection by briefly characterising the different agent types that can be derived as instances from the taxonomy. There are six possible combinations of agents:

1. *Autonomous hardware agents* (*H-AU*): these agents are characterised by the requirement for robust control within a physical environment, interleaving higher-level control and lower-level execution, coping with the unexpected in real time, and making up for the limitations of practical sensors and effectors (incomplete, erroneous knowledge). Most autonomous control systems shown in Table 1 fall into this category, e.g., RAPs, NMRA, and AuRA.
2. *Autonomous software agents* (*S-AU*): software that acts autonomously and makes decisions in a simulated or real software environment. An example is a software agent associated with a workflow. This agent autonomously plans and monitors the routing of the different tasks to be performed as part of the workflow. Also, autonomous software agents are often the back-end of what appears to be a software assistant agent (see below) in the front-end perspective. Thus, a system such as ShopBot (Grand et al., 1997) could be seen as consisting of a front-end software assistant agent maintaining the user profile, preprocessing user input and presenting results, and of an autonomous software agent that goes shopping in the Internet.
3. *Hardware assistant agents* (*H-AS*): hardware agents whose primary task it is to assist human users. One class of these agents are household robots (von Puttkamer et al., 1991). From the architectures considered in this survey, only $_3$T was used (among others) for human assistance purposes, e.g., as a wheelchair and for running errands. However, we suggest to consider another group of hardware agents for this class, i.e., those agents that interact with humans for entertainment or educational purposes. We are likely to see examples of this species in the form of smarter, more human-like and more interesting Tamagotchi-like hardware pets.
4. *Software assistant agents* (*S-AS*): programs that assist a human on the computer screen or in Personal Digital Assistants (PDAs), that act on behalf of that human, or that entertain the human. As illustrated in the ShopBot example above, software assistant agents are often used as front-end to a system the back-end of which are autonomous software agents. The main requirements software assistant agents have to satisfy are maintaining a user profile and adapting it to reflect changing user preferences, and find information that is relevant to a human according to her profile, and to present this information in a personalized way, i.e., in a way that is appropriate to the knowledge state of the user according to the profile. Note that entertainment agents such as Creatures (Grand et al., 1997), as well as educational agents (Johnson & Rickel, 1997) would fall into this category as well, according to the above definition.
5. *Hardware multiagents* (*H-MA*): hardware agents that act as entities in a multiagent system, e.g., cooperating robots in a manufacturing environment. Building hardware multiagents combines classical robotics requirements (see above) with the ability to reason about other agents, to form teams, and to perform joint plans and actions, e.g., in order to recognize and resolve goal conflicts or possibilities/necessities to cooperate to achieve a local or global goal.
6. *Software multiagents* (*S-MA*): programs that act as entities in a multiagent system. The most common application areas for software multiagents are the solution of dynamic and distributed resource allocation problems, as well as cooperative expert systems applications.

For each agent architecture considered in this paper, the third column of Table 1 shows what types of agents were built using this architecture.

7.3 Choosing the right agent to do the right thing

So far, we have described a set of agent architectures and defined a taxonomy of types of agents for

different applications. In this section, we address the question: *What is the right agent architecture to apply to a specific problem?*

Frankly, there is no black-or-white, deterministic answer to this. More often than not, the answer will be pre-determined by external factors (e.g., commercial availability, availability of tools and development environments, compliance with internal information infrastructure), and the choice of the agent architecture will be one of the smaller problems to be solved. For this reason, we provide some guidelines that may help a user decide which architecture to choose for a specific application domain.

At present, only a handful of commercial agent applications are available, most of which we discussed in section 6. Most of the architectures that were reviewed in this paper are research prototypes. Thus, if the reader intends to find an off-the-shelf solution, the choice will be restricted. The guidelines we provide below in particular aim at two groups of users: academic users who may adapt/extend an existing system to meet their specific requirements, and commercial users who see a potential in developing agent technology for a specific class of applications and who wish to make the right technological base decisions.

Guideline 1 Check carefully whether you need agents, or whether another programming paradigm, such as (distributed) objects, will do the job to solve your application problem. Be requirements-driven rather than technology-driven. If your application problem shows some of the following properties, then you might want to consider using agents:

- highly dynamic, necessary to be responsive and adaptable to a changing environment;
- need to deal with failure, e.g., re-scheduling, re-planning, re-allocating of resources;
- need to balance long-term, goal-directed and short-term, reactive behaviour;
- complex and/or safety-critical, guaranteed reaction and response times;
- geographically or logically distributed, autonomous or heterogeneous nodes;
- need for reliability, robustness, and maintainability;
- complex or decentralised resource allocation problems with incomplete information;
- flexible interaction with human users.

For instance, imagine your task is to build a workflow management system to improve your company's information processes. *If* the business processes in your company are well-understood, largely involve information flow but no material flow, *if* there are clear and well-established ways of dealing with failure, *if* the services provided by different departments are static and known *a priori*, *then* you might as well model your workflow management system as a distributed object-oriented application.

If, however, the set of services is expected to change or service-level agreements are likely to be negotiable depending on the requestor of a workflow service, *if* workflows are dynamic and need to be completed within a short period of time, and *if* your workflow system is likely to cater for the needs of offline workers and has to scale up to workflow processes beyond the control of an individual enterprise, *then* consider using agent technology, possibly based on a distributed object-oriented approach.

Guideline 2 Table 1 provides a basic guideline for which architectures are potentially useful to deal with which class of application. The fact that architecture A was successfully used to build applications of class P, but never used to build applications of class Q does not necessarily mean that A is not appropriate to deal with Q; however, if your problem is P, at least you have some positive evidence that A will work. Thus, use Table 1 for a rough orientation.

Guideline 3 If your problem requires autonomous hardware agents, then chances are that you are well served with one of the hybrid architectures. A purely reactive approach may be applicable if you can find a decomposition of your system that allows you to define very simple agents. But do not underestimate the difficulty of achieving meaningful self-organising behaviour from a set of simple agents.

Guideline 4 If your problem requires autonomous software agents, then you can choose between some robust architectures such as dMARS and SOAR. dMARS is a commercial product, and hence not available for free; however, there are various implementations of PRS, one of which, UM-PRS, was developed at Michigan University and to our knowledge is available for free.

Guideline 5 If your problem requires software assistant agents, then the agent architecture is definitely not the first thing to worry about. What is much more important is to get the domain functionality such as profiling and personalisation right. These, however, are much more problems of human–computer interaction (HCI), user modelling and pattern matching.

Guideline 6 If your problem requires hardware assistant agents, then there is no off-the-shelf system available. A solution to your problem may be to select any architecture for ACSs (see also Guideline 3) and extend it by adding the required HCI functionality (see Guideline 5).

Guideline 7 If your problem requires software multiagents, then you might want to look at any of the examples presented under the *Interacting Agents* category. However, if there are high interoperability requirements on your system (e.g., communication with non-agent components within your company), then you may run into trouble as none of the systems described easily complies with interoperability standards such as CORBA. You therefore may have to modify the communication layer. When doing so, make sure that you do not do double work: there are well-established means to transport a stream of bytes from one place to another. What is missing is a system that can deal with the semantics of these byte streams, and this is where agents can help.

Guideline 8 If your problem requires hardware multiagents, then either select one of the architectures or systems for autonomous hardware agents (see Guideline 3) and add your cooperation knowledge to these, or select one of the cooperation-centred architectures presented in section 4 and enhance them by the necessary interface to your hardware. An architecture like INTERRAP might be of interest for you as it has been applied to the domain of interacting robots. However, its current status is that of a research prototype.

Guideline 9 Do not break a butterfly on a wheel. While it is appealing to compare an Internet search agent with a robot, this analogy must not be taken too literally. Most architectures that are used to control robots are far too heavy-weight. If the domain you are working in is a software domain, your architecture of choice should:

- be capable of multi-tasking;
- decouple low-level message handling from high-level message interpretation;
- come with a service model allowing agents to vend services that are internally mapped into tasks;
- comply with interoperability standards such as CORBA to make agent services available to a wide spectrum of applications;
- have a small footprint: the empty agent should not be bigger than a few 100K.

Guideline 10 For most interesting applications, neither purely reactive nor purely deliberative architectures are useful. The odds are that you are best served with one of the hybrid architectures. Systems such as PRS, RAPS or SOAR have been around for years and are sufficiently stable and mature to be a good choice for any type of autonomous control systems.

Guideline 11 If adaptability is crucial to solve your application problem, you will not have much choice. Most research reviewed in this paper has neglected the ability of an agent to learn, and it is not clear how the architectures could support enabling an agent to deal with longer term change. A notable exception is the SOAR system, so you might want to have a look at that. Some approaches, in particular the reactive agent approaches, provide a short-term form of adaptability based on feedback. Also, Ferguson's Touring Machines provide a modelling layer that can be used to change planning parameters. Also, learning a user's preferences is an important issue in assistant agents applications. However, to our knowledge none of the described architectures offers a uniform and complete model for adaptability.

7.4 Discussion

The first aspect of the discussion is the taxonomy itself. The reader may wonder why we defined our own taxonomy instead of adopting e.g., either of Franklin and Graesser (1997) or Nwana (1996). Also, why did we introduce two different schemes of classification in the two parts of this paper?

To answer the first part of this question: the Franklin–Graesser taxonomy has been a general, rather philosophical attempt to structure the field. As such, it does not reflect the view of a system designer who wishes to apply agent technology to a specific application domain. On the other hand, Nwana's taxonomy is restricted to software agents. Therefore, we preferred not to adopt any of the existing taxonomies.

With regard to the second part of the question, it is important to note that what we classified in the first part of this paper (reactive, deliberative, interacting, hybrid, others), were agent architectures. However, the classification proposed in section 7.2 refers to agents or agent systems that were built according to an agent architecture. Thus, the two schemes classify different entities. Note that different agent types may appear in one and the same application. The most important requirement to be satisfied by the latter scheme is that it can be used to classify specific instances of agents built according to a specific agent architecture, and used in a specific class of applications.

The second topic of discussion relates to the guidelines. Clearly, they are rules of thumb, and should be treated as such. They are a result of analysing current agent architectures and the applications they were used for, as well as of our experience in designing agent architectures and agent-based systems. Guidelines 2 to 8 are more or less directly derived from Table 1. The guidelines reflect the current state of the art and are prone to changes. For instance, Guideline 9 seems to suggest to use "lighter" architectures for autonomous software agent applications than for autonomous hardware agents. In fact, what it does suggest is to make this choice *given the current state of the art in agent research*, based on observing the application areas of currently existing architectures, as well as on experiences with two architectures that we developed in the past and applied to hardware and software domains alike.

Bridging the current gap, i.e., materialising the intuitive analogy between robots and softbots in terms of architectures that can be used to deal with the common aspects of both, is an interesting topic for future research.

Similarly, Guideline 11 should be understood as a requirement for future research on integrating learning mechanisms into agent architectures. There are plenty of applications that require only a limited (short-term) form of learning which is provided by at least some systems that are available today. What is lacking is longer-term adaptability.

8 Conclusion

The main contribution of this paper is twofold: first, it offers a survey of agent control architectures, aimed at giving an accessible overview of the field to readers not familiar with agent technology. Secondly, it aims at providing the system designer with some guidelines as to which type of architecture is best suited for which type of application.

Given the variety of the approaches described in this paper, one might wonder whether there is some sort of convergence towards a generally accepted agent architecture. One quick and simple answer is that, unfortunately, this is not the case: researchers are still debating the definition of an agent (see Müller et al. (1997, Part II) for a good survey of this debate), and a general agent architecture does not seem to be in sight, as long as there is no general agreement on the basics of an agent. A more optimistic answer, however, is that despite the ongoing debate, there are a variety of architectures that constitute operational models for implemented agent languages. Layered agent architectures and BDI architectures are widely agreed upon, and there is active research work on combining them (Fischer et al., 1996; Kinny & Georgeff, 1997). Design methodologies for agents and agent-based systems are becoming a research issue in their own right.

The ongoing discussion about a unique and generally accepted definition of agents and agent architectures is somewhat similar to the discussion of what programming language would be the

Table 1 Agent architectures and agent applications

Architecture	Applications	Classification	Reference
Reactive agents			
Subsumption architecture	mobile robots and land vehicles	H-AU	(Brooks, 1986)
Self-organising agents	mobile robots, emerging group behaviour	H-AU, (H-MA)	(Steels, 1990)
AuRA	mobile robots and land vehicles	H-AU, (H-MA)	(Arkin, 1990)
Dynamic action selection	technique has been used in mobile robots and Artificial Life applications	H-AU	(Maes, 1989)
PENGI	arcade computer game	S-AU	(Agre & Chapman, 1990)
ECO model	distributed problem-solving	S-MA	(Ferber, 1989)
Deliberative agents			
IRMA	general architecture, probably robotics	H-AU	(Bratman, Israel & Pollack, 1987)
PRS	originally: robotics	H-AU	(Georgeff & Ingrand, 1989)
dMARS	air traffic control, business process enactment and monitoring	H-AU, S-AU, S-AS, (S-MA)	(Rao & Georgeff, 1995)
SOAR	general AI problem-solving architecture; mainly autonomous control systems, more recently also believable agents	H-AU, S-AU, S-AS	(Laird, Newell & Rosenbloom, 1987; Tambe et al., 1994)
Cypress	mobile robots and vehicles	H-AU	(Wilkins, Myers & Wesley, 1994)
Agent0/PLACA	no specific application mentioned, supports communicative acts	H-MA, S-MA	(Shoham, 1993; Thomas, 1995)
Interacting agents			
MAGSY	production planning, distributed resource allocation, cooperating expert systems	S-MA, S-AS	(Fischer, 1993)
GRATE*	electricity network diagnosis, later: workflow management	S-MA, S-AS	(Jennings, 1992b; Jennings et al., 1996; Norman et al., 1997)
MECCA	traffic control systems, personal digital assistants	S-AS, S-MA	(Steiner et al., 1993)
COSY	production planning, transport planning	S-MA	(Burmeister & Sundermeyer, 1992)

Hybrid architecture			
RAPs, ATLANTIS, $_3$T	mobile robots and land vehicles, robotics wheelchair, errand running ($_3$T)	H-AU, (H-AS)	(Firby, 1989; Gat, 1991b; Bonasso et al., 1996)
Lyons and Hendriks	mobile robots and land vehicles	H-AU	(Lyons & Hendriks, 1992)
TouringMachines	mobile robots and land vehicles	H-AU, (S-AU)	(Ferguson, 1992)
INTERRAP	cooperating robots, flexible transport systems, game playing agents, logistics	H-MA, (S-MA)	(Müller, 1996b)
SimAgent	humanoid robots or softbots (very abstract architecture)	H-AU, S-AU	(Sloman & Poli, 1996)
NMRA	spacecraft control	H-AU	(Pell et al., 1997)
Software agents			
Tok	believable agents for entertainment and arts applications	S-AS	(Reilly, 1996)
VET	education and training	S-AS	(Johnson & Rickel, 1997)
ShopBot	resource discovery, electronic commerce	S-AS, S-AU	(Grand et al., 1997)
Zuno VRISKO, QuarterDeck WebCompass, AgentSoft LifeAgent Pro, Firefly ...	information retrieval and information filtering, personalisation	S-AU, S-AS	—

best to be used. This analogy might suggest that there is no single *best* agent architecture: as in the case of programming languages, the best choice depends upon the type of application to be modelled and the problem to be solved. Section 7 complies with this view by proposing an application-related taxonomy for agents and by offering some guidelines for relating agent architectures to agent applications.

Also, whereas there is a strong need for standardisation of agent communication languages, such a need is not directly obvious for agent architectures: if scientific initiatives (FIPA, DARPA Knowledge Sharing Effort) or industrial standards bodies (OMG, W3C) succeed in developing uniform interfaces and communication standards that allow different and possibly heterogeneous agents to interact, we can live with different definitions, theories, architectures and languages for describing individual agents for quite a while.

Acknowledgements

I am grateful to the two anonymous reviewers for their invaluable and constructive comments. I also would like to thank Michael Wooldridge, Nick Jennings, Markus Pischel, and Innes Ferguson for many fruitful discussions about agent architectures and agent applications that shaped my current view. Thanks to Eva Völker for careful proof-reading. Of course, I am fully responsible for any remaining flaws.

Appendix A: Agent architectures, types and applications

Table 1 overviews the architectures described in this paper (first column), the main types of applications that were built using these architectures (second column), and the corresponding classification of the agents built using these architectures according to section 7.2 (third column), using the abbreviations defined in that section. For each architecture mentioned, at least one reference is given in the fourth column of the table. Table 1 contains some architectures or systems that were not described in detail, but that were influential (e.g., SOAR) or interesting for some other reasons.

References

Agentsoft home page. `http://www.agentsoft.com`.

Agre, PE and Chapman, D, 1987. "Pengi: an implementation of a theory of activity" *Proc. of AAAI-87* Morgan Kaufmann, pp. 268–272.

Agre, PE and Chapman, D, 1990. "What are plans for?" in Maes, P (ed), *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back* MIT/Elsevier, pp. 17–34.

Allen, JF, Hendler, J and Tate, A, 1990. *Readings in Planning* Morgan Kaufmann.

Ambros-Ingersson, JA and Steel, S, 1990. "Integrating planning, execution, and monitoring" in Allen, JF, Hendler, J and Tate, A (eds), *Readings in Planning* Morgan Kaufmann, pp. 735–740.

Arkin, RC, 1990. "Integrating behavioral, perceptual, and world knowledge in reactive navigation" in Maes, P (ed), *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back* MIT/Elsevier, pp. 105–122.

Babloyantz, A, 1986. *Molecules, Dynamics and Life. An Introduction to Self-Organization of Matter* Wiley.

Balch, T and Arkin, RC, 1995. "Motor schema-based formation control for multiagent robot teams" *Proc. First International Conference on Multiagent Systems*.

Barbuceanu, M and Fox, MS, 1996. "The architecture of an agent building shell" in Wooldridge, M, Müller, JP and Tambe, M (eds), *Intelligent Agents – Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95), volume 1037 of Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 235–250.

Bates, J, Loyall, AB and Reilly, WS, 1992. "An architecture for action, emotion, and social behavior" *Proc. Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-92)*.

Bates, J, 1994. "The role of emotions in believable agents" *Communications of the ACM* **37**(7) 122–125.

Bonasso, RP, Kortenkamp, D, Miller, DP and Slack, M, 1996. "Experiences with an architecture for intelligent,

reactive agents'' in Wooldridge, M, Müller, JP and Tambe, M (eds), *Intelligent Agents – Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95), volume 1037 of Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 187–202.

Bond, A and Gasser, L, 1988. *Readings in Distributed Artificial Intelligence* Morgan Kaufmann.

Bratman, ME, Israel, DJ and Pollack, ME, 1987. ''Toward an architecture for resource-bounded agents'' Technical Report CSLI-87-104, Center for the Study of Language and Information, SRI and Stanford University.

Bratman, ME, 1987. *Intentions, Plans, and Practical Reason* Harvard University Press.

Brooks, RA, 1986. ''A robust layered control system for a mobile robot'' *IEEE J. Robotics and Automation* **2**(1) 14–23.

Brooks, RA, 1990. ''A robot that walks: Emergent behaviors from a carefully evolved network'' in Winston, PH and Shellard, SA (eds), *Artificial Intelligence at MIT, Expanding Frontiers* MIT Press, pp. 28–39.

Brooks, RA, 1991. ''Intelligence without representation'' *Artificial Intelligence* **47** 139–159.

Bürckert, HJ and Müller HJ, 1991. ''RATMAN: Rational Agents Testbed for Multi-Agent Networks'' in Demazeau, Y and Müller JP (eds), *Decentralized A. I., volume 2*, North-Holland, pp. 217–230.

Burmeister, B and Sundermeyer, K, 1992. ''Cooperative problem-solving guided by intentions and perception'' in Demazeau, Y and Werner, E (eds), *Decentralized A. I., volume 3*, North-Holland.

Burmeister, B, Haddadi, A and Sundermeyer, K, 1993. ''Generic configurable cooperation protocols for multi-agent systems'' *Pre-Proceedings of MAAMAW93* University of Neuchâtel.

Chaib-Draa, B, 1996. ''Interaction between agents in routine, familiar and unfamiliar situations'' *Int. J. Intelligent and Cooperative Information Systems* (to appear).

Cohen, P and Levesque, H, 1990. ''Intention is choice with commitment'' *Artificial Intelligence* **42**(3).

Cohen, PR, Greenberg, ML, Hart, DM and Howe, AE, 1989. ''Trials by fire: Understanding the design requirements for agents in complex environments'' *AI Magazine* **10**(3).

Conway, J, 1976. *On Numbers and Games* Academic Press.

Dabija, VG, 1993. ''Deciding Whether to Plan to React'' PhD Dissertation, Department of Computer Science, Stanford University.

Davis, R and Smith, RG, 1983. ''Negotiation as a metaphor for distributed problem solving'' *Artificial Intelligence* **20** 63–109.

Davis, DN, 1997. ''Reactive and motivational agents: towards a collective minder'' in Müller, JP, Wooldridge, MJ and Jennings, NR (eds), *Intelligent Agents III – Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96), Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 309–324.

Dean, TL and Wellman, MP, 1991. *Planning and Control* Morgan Kaufmann.

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

Drummond, M and Bresina, J, 1990. ''Anytime synthetic projection: Maximizing the probability of goal satisfaction'' *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)* AAAI Press/MIT Press, pp. 138–144.

Drummond, ME and Kaelbling, LP, 1990. ''Integrated agent architectures: Benchmark tasks and evaluation metrics'' *Proceedings DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control* Morgan Kaufmann, pp. 408–411.

Emerson, EA and Srinivasan, J, 1989. ''Branching time temporal logic'' in de Bakker, JW, de Roever, WP and Roezenberg, G (eds), *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency* Springer-Verlag, pp. 123–172.

Etzioni, O and Weld, D, 1994. ''A softbot-based interface to the internet'' *Communications of the ACM* **37**(7) 72–76.

Etzioni, O, 1996. ''Moving up the information food chain: Deploying softbots on the world wide web'' *Proceedings of AAAI-96 (Abstract of invited talk)*.

Extempo Systems Inc., 1997. ''Improvisational characters: Bringing interactive products to life'' Extempo Systems Inc.: `http://www.extempo.com/webBar`.

Ferber, J, 1989. ''Eco-problem solving: How to solve a problem by interactions'' *Proc. 9th Workshop on DAI* pp. 113–128.

Ferguson, IA, 1992. ''TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents'' PhD Dissertation, Computer Laboratory, University of Cambridge, UK.

Ferguson, IA, 1995. ''Integrated control and coordinated behaviour'' in Wooldridge, MJ and Jennings, NR (eds), *Intelligent Agents – Theories, Architectures, and Languages, volume 890 of Lecture Notes on AI.* Springer-Verlag.

Fikes, RE, Hart, PE and Nilsson, N, 1971. ''STRIPS: A new approach to the application of theorem proving'' *Artificial Intelligence* **2** 189–208.

Finin, T and Fritzson, R, 1994. ''KQML – a language and protocol for knowledge and information exchange'' *Proceedings of the 13th Intl. Distributed Artificial Intelligence Workshop* pp. 127–136.

Firby, RJ, 1989. "Adaptive Execution in Dynamic Domains" PhD Dissertation, Yale University, Computer Science Department. (Also published as Technical Report YALEU/CSD/RR#672.)

Firby, RJ, 1992. "Building symbolic primitives with continuous control routines" in Hendler, J (ed), *Proc. 1st International Conference on Artificial Intelligence Planning Systems (AIPS-92)* Morgan Kaufmann.

Firby, RJ, 1994. "Task networks for controlling continuous processes" *Proc. 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)* pp. 49–54.

Fischer, K, Müller, JP and Pischel, M, 1996. "A pragmatic BDI architecture" in Wooldridge, M, Müller, JP and Tambe, M (eds), *Intelligent Agents – Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95), volume 1037 of Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 203–218.

Fischer, K, 1993. "Verteiltes und kooperatives Planen in einer flexiblen Fertigungsumgebung" DISKI, Dissertationen zur Künstlichen Intelligenz, infix.

Franklin, S and Graesser, A, 1997. "Is it an agent, or just a program?: A taxonomy for autonomous agents" in Müller, JP, Wooldridge, MJ and Jennings, NR (eds), *Intelligent Agents III – Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96), Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 21–36.

Frost, HR, 1996. "The JAVA agent template" `http://cdr.stanford.edu/ABE/JavaAgent.html`.

Gasser, L and Huhns, M, 1989. *Distributed Artificial Intelligence, Volume II* Research Notes in Artificial Intelligence, Morgan Kaufmann.

Gat, E, 1991a. "Alfa: a language for programming reactive robotic control systems" *Proc. IEEE Conference on Robotics and Automation*.

Gat, E, 1991b. "Reliable Goal-directed Reactive Control for Real-World Autonomous Mobile Robots" PhD Dissertation, Virginia Polytechnic and State University, Blacksburg, Virginia.

Gat, E, 1992. "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots" *Proc. AAAI'92* pp. 809–815.

Genesereth, MR and Ketchpel, SP, 1994. "Software agents" *Communications of the ACM* **37**(7) 48–53.

Georgeff, MP and Ingrand, FF, 1989. "Decision-making in embedded reasoning systems" *Proc. 6th International Joint Conference on Artificial Intelligence* pp. 972–978.

Georgeff, MP and Lansky, AL, 1986. "Procedural knowledge" *Proc. IEEE Special Issue on Knowledge Representation* **74** 1383–1398.

Georgeff, M, 1983. "Communication and interaction in multi-agent plans" *Proc. IJCAI-83* pp. 125–129.

Gmytrasiewicz, PJ, 1996. "On reasoning about other agents" in Wooldridge, M, Müller, JP and Tambe, M (eds) *Intelligent Agents – Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95), volume 1037 of Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 143–155.

Grand, S, Cliff, D and Malhotra, A, 1997. "Creatures: Artificial life autonomous software agents for home entertainment" in Johnson, WL (ed), *Proc. First International Conference on Autonomous Agents* ACM, pp. 22–29.

Haddadi, A, 1996. *Communication and Cooperation in Agent Systems: A Pragmatic Theory, volume 1056 of Lecture Notes in Artificial Intelligence* Springer-Verlag.

Hayes-Roth, B and van Gent, R, 1997. "Story-making with improvisational puppets" *Proc. First International Conference on Autonomous Agents* ACM Press, pp. 1–7.

Hayes-Roth, B, 1990. "Architectural foundations for real–time performance in intelligent agents" *Real-Time Systems: The International Journal of Time-Critical Computing Systems* **2** 99–125.

Hayes-Roth, B, 1995. "An architecture for adaptive intelligent systems" *Artificial Intelligence* **72** 329–365.

Huhns, MN and Singh, MP (eds), 1998. *Readings in Agents* Morgan Kaufmann.

Jennings, NR, Faratin, P, Johnson, MJ, Norman, TJ, O'Brien, P and Wiegand, ME, 1996. "Agent-based business process management" *International Journal of Cooperative Information Systems* **5**(2&3) 105–130.

Jennings, NR, 1992a. "Joint Intentions as a Model of Multi-Agent Cooperation" PhD Dissertation, Queen Mary and Westfield College, London.

Jennings, NR, 1992b. "Towards a cooperation knowledge level for collaborative problem solving" *Proc. 10th European Conference on Artificial Intelligence* pp. 224–228.

Johnson, WL and Rickel, J, 1997. "Integrating pedagocial capabilities in a virtual environment agent" *Proc. First International Conference on Autonomous Agents* ACM Press, pp. 30–38.

Johnson, WL (ed), 1997. *Proceedings of the First International Conference on Autonomous Agents* ACM Press.

Kaelbling, LP and Rosenschein, SJ, 1990. "Action and planning in embedded agents" in Maes, P (ed), *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back* MIT/Elsevier, pp. 35–48.

Kaelbling, LP, 1990. "An architecture for intelligent reactive systems" in Allen, J, Hendler, J and Tate, A (eds), *Readings in Planning* Morgan Kaufmann, pp. 713–728.

Kinny, D and Georgeff, MP, 1991. "Commitment and effectiveness of situated agents" *Proc. Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)* pp. 82–88.

Kinny, DN and Georgeff, MP, 1997. "Modelling and design of multi-agent systems" in Müller, JP, Wooldridge,

MJ and Jennings, NR (eds), *Intelligent Agents III – Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96), Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 1–20.

Kinny, D, Ljungberg, M, Rao, A, Sonenberg, E, Tidhar, G and Werner, E, 1992. "Planned team activity" in Cesta, A, Conte, R and Miceli, M (eds), *Pre-Proceedings of MAAMAW'92*.

Kinny, D, Georgeff, MP and Rao, AS, 1996. "A methodology and modelling technique for systems of BDI agents" in van de Velde, W and Perram, JW (eds), *Agents Breaking Away – 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), volume 1038 of Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 56–71.

Kobsa, A and Wahlster, W (eds), 1989. *User Models in Dialog Systems* Springer-Verlag.

Konolige, K, 1997. "COLBERT: A language for reactive control in saphira" in *Proceedings of the German Conference on Artificial Intelligence (KI-98)*.

Laird, JE, Newell, A and Rosenbloom, PS, 1987. "SOAR: an architecture for general intelligence" *Artificial Intelligence* **33**(1) 1–62.

Lange, DB and Chang, DT, 1996. "IBM aglets workbench: Programming mobile agents in Java" `http://www.trl.ibm.co.jp:80/aglets/whitepaper.htm`.

Langton, CG, 1989. "Artificial life" in Langton, CG (ed), *Artificial Life* Addison-Wesley.

Lux, A and Steiner, DD, 1995. "Understanding cooperation: an agent's perspective" *Proc. First International Conference on Multiagent Systems*.

Lux, A, 1995. "Kooperative Mensch-Maschine Arbeit – ein Modellierungsansatz und dessen Umsetzung im Rahmen des Systems MEKKA" PhD Dissertation, Universität des Saarlandes, Saarbrücken.

Lyons, DM and Hendriks, AJ, 1992. "A practical approach to integrating reaction and deliberation" *Proc. 1st International Conference on AI Planning Systems (AIPS)* Morgan Kaufmann, pp. 153–162.

Maes, P, 1989. "The dynamics of action selection" *Proc. IJCAI-89* pp. 991–997.

Maes, P (ed), 1990a. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back* MIT/Elsevier.

Maes, P, 1990b. "Situated agents can have goals" in Maes, P (ed), *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back* MIT/Elsevier, pp. 49–70.

Maes, P, 1994a. "Agents that reduce work and information overload" *Communications of the ACM* **37**(3) 31–40.

Maes, P, 1994b. "Modeling adaptive autonomous agents" *Artificial Life Journal* **1**(1 & 2).

Mataric, M, 1993. "Synthesizing group behaviors" *Proc. IJCAI Workshop on Dynamically Interacting Robots* pp. 1–10.

Mayfield, J, Labrou, Y and Finin, T, 1996. "Evaluating KQML as an agent communication language" in Wooldridge, M, Müller, JP and Tambe, M (eds), *Intelligent Agents – Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95), volume 1037 of Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 347–360.

McCabe, FG and Clark, KL, 1995. "April—agent process interaction language" in Wooldridge, MJ and Jennings, NR (eds), *Intelligent Agents – Theories, Architectures, and Languages, volume 890 of Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 324–340.

McDermott, D, 1991. "Robot planning" Technical Report 861, Yale University, Department of Computer Science.

Meyer, J-JC, van der Hoek, W and Vreeswijk, GAW, 1991. "Epistemic logic for computer science: A tutorial (part one)" *Bulletin of the EATCS* **44** 242–270.

Minsky, M, 1986. *The Society of Mind* Simon and Schuster (Touchstone).

Müller, JP and Pischel, M, 1994. "An architecture for dynamically interacting agents" *International Journal of Intelligent and Cooperative Information Systems (IJICIS)* **3**(1) 25–45.

Müller, JP, Wooldridge, MJ and Jennings, NR (eds), 1997. *Intelligent Agents III – Agent Theories, Architectures, and Languages, volume 1193 of Lecture Notes in Artificial Intelligence* Springer-Verlag.

Müller, JP, 1996a. "A cooperation model for autonomous agents" *Proc. Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*.

Müller, JP, 1996b. *The Design of Autonomous Agents – A Layered Approach, volume 1177 of Lecture Notes on Artificial Intelligence* Springer-Verlag.

Newell, A and Simon, HA, 1976. "Computer science as empirical enquiry: Symbols and search" *Communications of the ACM* **19**(3) 113–126.

Nicolis, G and Prigogine, I, 1977. *Self-organization in Non-equilibrium Systems* Wiley.

Norman, TJ and Long, D, 1996. "Alarms: An implementation of motivated agency" in Wooldridge M, Müller, JP and Tambe, M (eds), *Intelligent Agents – Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95), volume 1037 of Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 219–234.

Norman, TJ, Jennings, NR, Faratin, P and Mamdani, EH, 1997. "Designing and implementing a multi-agent architecture for business process management" in Müller, JP, Wooldridge, MJ and Jennings, NR (eds),

Intelligent Agents III – Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96), Lecture Notes on Artificial Intelligence Springer-Verlag, pp. 261–276.

Nwana, HS, 1996. "Software agents: an overview" *Knowledge Engineering Review* **11**(3) 205–244.

Pell, B, Bernhard, DE, Chien, SA, Gat, E, Muscettola, N, Nayak, PP, Wagner, MD and Williams, BC, 1997. "An autonomous spacecraft agent prototype" in Johnson, WL (ed), *Proc. First International Conference on Autonomous Agents* ACM, pp. 253–261.

Petrie, C, 1997. "What is an agent?" in Müller, JP, Wooldridge, MJ and Jennings, NR (eds), *Intelligent Agents III – Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96), Lecture Notes on Artificial Intelligence* Springer-Verlag, pp. 41–44.

Rao, AS and Georgeff, MP, 1991a. "Modeling agents within a BDI-architecture" in Fikes, R and Sandewall, E (eds), *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)* Morgan Kaufmann, pp. 473–484.

Rao, AS and Georgeff, MP, 1991b. "Modeling rational agents within a BDI-architecture" Technical Report 14, Australian AI Institute, Carlton, Australia.

Rao, AS and Georgeff, MP, 1992. "An abstract architecture for rational agents" *Proc. 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)* Morgan Kaufmann, pp. 439–449.

Rao, AS and Georgeff, MP, 1995. "BDI-agents: from theory to practice" *Proc. First Intl. Conference on Multiagent Systems*.

Reilly, WSN, 1996. "Believable Social and Emotional Agents" PhD Dissertation, School of Computer Science, Carnegie Mellon University.

Rosenschein, JS and Zlotkin, G, 1994. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers* MIT Press.

Russell, S and Norvig, P, 1995. *Artificial Intelligence: A Modern Approach* Prentice Hall.

Sacerdoti, ED, 1975. "The nonlinear nature of plans" in *IJCAI-75* pp. 206–218.

Sanborn, J and Hendler, J, 1990. "A model of reaction for planning in dynamic environments" *International Journal of Artificial Intelligence in Engineering* **6**(1) 41–60.

Shoham, Y, 1993. "Agent-oriented programming" *Artificial Intelligence* **60** 51–92.

Simon, HA, 1981. *The Sciences of the Artificial* MIT Press.

Sloman, A and Poli, R, 1996. "SIM_AGENT: A toolkit for exploring agent design" in Wooldridge M, Müller, JP and Tambe, M (eds), *Intelligent Agents – Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95), volume 1037 of Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 392–407.

Steels, L, 1990. "Cooperation between distributed agents through self-organization" in Demazeau, Y and Müller, JP (eds), *Decentralized A.I* North-Holland, pp. 175–196.

Steiner, DD, Burt, A, Kolb, M and Lerin, C, 1993. "The conceptual framework of MAI^{2}L" *Pre-Proceedings of MAAMAW'93*.

Straßer, M, Baumann, J and Hohl, F, 1996. "Beyond JAVA: Merging CORBA-based mobile agents and WWW" *Joint W3C/OMG Workshop on Distributed Objects and Mobile Code (Accepted Position Paper)*.

Suchman, LA, 1987. *Plans and Situated Actions* Cambridge University Press.

Tambe, M, Jones, R, Laird, JE, Rosenbloom, PS and Schwamb, KB, 1994. "Building believable agents for simulation environments" *Proc. AAAI Spring Symposium: Believable Agents* AAAI.

Thomas, SR, 1993. "PLACA, an Agent Oriented Programming Language" PhD Dissertation, Stanford University. (Available as Stanford University Computer Science Department Technical Report STAN-CS-93-1487.)

Thomas, SR, 1995. "The PLACA agent programming language" in Wooldridge, MJ and Jennings, NR (eds), *Intelligent Agents – Theories, Architectures, and Languages, volume 890 of Lecture Notes in Artificial Intelligence* Springer-Verlag, pp. 355–370.

Vardi, MY, 1986. "On epistemic logic and logical omniscience" *Proc. First Conference on Theoretical Aspects of Reasoning about Knowledge (TARK'86)* Morgan Kaufmann, pp. 293–306.

von Puttkamer, E, Tjutjunikow, I and Trieb, R, 1991. "Local obstacle avoidance and acceleration based motion control for autonomous mobile robots" *Proc. 2nd Workshop on Sensor Fusion and Environmental Modelling* IARP – International Advanced Robotics Programme.

Wilkins, DE, Myers, KL and Wesley, LP, 1994. "Cypress: Planning and reacting under uncertainty" in Burstein, MH (ed), *ARPA/Rome Laboratory Planning and Scheduling Initiative Workshop Proceedings* Morgan Kaufmann, pp. 111–120.

Wilkins, DE, 1988. *Practical Planning: Extending the Classical AI Planning Paradigm* Morgan Kaufmann.

Wooldridge, MJ and Jennings, NR (eds), 1995. *Intelligent Agents – Theories, Architectures, and Languages, volume 890 of Lecture Notes in Artificial Intelligence* Springer-Verlag.

Wooldridge M, Müller, JP and Tambe, M (eds), 1996. *Intelligent Agents II, volume 1037 of Lecture Notes in Artificial Intelligence* Springer-Verlag.