The Right Agent (Architecture) to Do the Right Thing

Jörg P. Müller*

John Wiley & Sons, Inc.

Abstract. Academic and industrial system designers who consider using agent technology to solve an application problem are faced with a wide variety of agent paradigms: There are deliberative agents, reactive agents, interacting agents, hybrid agents, layered agents, believable agents, mobile agents, software agents, softbots — the list could well be prolonged. Also, within each paradigm, the user can select between different architectures and systems, making the actual choice a complex and difficult endeavor.
The objective of this paper is to assist readers in deciding which agent architecture to choose for a specific *application*. We approach this objective in three steps. First, 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 the first step, we propose a classification of agents according to different classes of applications. Based on this classification, the third step is to provide 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.

1 Introduction

An important research branch in AI in the early nineties has investigated control architectures for intelligent agents. An agent architecture describes the functional components of an agent and 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, and building on a wide variety of research disciplines. Indeed the variety of agent architectures and systems is so wide that system designers in academia and industry, who are willing to "try" intelligent agents technology to solve an application problem, are often lost when it comes to decide what is the most suitable agent architecture for their specific problem.

In 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, our aim is to provide a set of guidelines that help the system designer select the right — i.e., most appropriate — architecture for a given problem domain. Thus, this paper is intended for system designers and software engineers interested in agent technology as a software-engineering paradigm.

Our approach in this paper is based on empirical evidence: we study existing agent architectures and the applications that were built using these architectures, as far as known from the literature. Based on this test set, we try to derive a taxonomy that

* Email: jpm@wis-dev.wiley.co.uk

J.P. Müller et al. (Eds.): ATAL'98, LNAI 1555, pp. 211-225, 1999.
Springer-Verlag Berlin Heidelberg 1999

classifies agents in terms of the classes of applications they appear in, and we define a set of guidelines that we hope will help system designers in identifying what is *the right agent to do the right thing*.

The paper is organized as follows: Section 2 presents the test set that we use, i.e., the set of example agent architectures. Section 3 identifies the core application for the architectures in the test set. In Section 4, we propose a taxonomy for agents based on the classes of applications identified in Section 3. A set of guidelines for choosing agent architectures is presented in Section 5. In Section 6, we discuss these guidelines and how to use them. The paper finishes with some concluding remarks in Section 7. Table 6 in Appendix A summarizes the analysis of the test set.

2 The Test Set

The test set used in this paper consists of ca. 30 agent architectures. It is an updated and extended version of the architectures described in an earlier review [26]. The architectures discussed vary widely in underlying models, approaches, and features. Available implementations range from proof-of-concept prototypes over research implementations to semi-commercial or commercial systems.

In accordance with [26] we classify our test set into five categories, i.e., *reactive agents*, *deliberative agents*, *interacting agents*, *layered approaches*, and *others*. The former four categories reflect different architectural paradigms (see [25] for a detailed discussion). The latter category serves as a container for various more recent approaches that do not fit nicely in any of the former, such as believable agents, softbots, and a variety of commercial agent-based systems.

A detailed description of the architectures in the test set would exceed the boundaries of this paper. We refer to the original article. In the remainder of this section, we provide a brief characterization of each category and its instances. For a complete list of the architectures contained in the test set, we refer to Table 6 in Appendix A.

The first category, *reactive agents*, include examples such as Brooks's subsumption architecture. Research on reactive agents is strongly influenced by behaviorist psychology and Artificial Life. Reactive agents make decisions based on a very limited amount of information, and simple situation-action rules. Some researchers denied the need of any symbolic representation of the world; instead, reactive agents make decisions directly based on sensory input. The focus of this class of system is directed towards achieving *robust* behavior instead of *correct* or *optimal* behavior. Table 1 shows the instances of reactive agents that we consider in this paper.

Deliberative agents are based on Simon and Newell's physical symbol system hypothesis 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. AI planning systems and BDI agents are the classical representatives of this category (see Table 2).

The third category, *interacting agents*, have their origin in Distributed Artificial Intelligence (DAI). DAI deals with coordination and cooperation among distributed intelligent agents. While the focus of this research discipline has been on the coordination process itself and on mechanisms for cooperation among autonomous agents rather

Architecture	Reference
Subsumption architecture	[6]
Self-organizing agents	[38]
AuRA	[2]
Dynamic action selection	[23]
PENGI	[1]
ECO model	[9]

Table 1. Reactive agents

Architecture	Reference
IRMA	[5]
PRS	[15]
dMARS	[33]
SOAR	[21] [41]
Cypress	[43]
Agent0 / PLACA	[35] [42]

Table 2. Deliberative agents

than on the structure of these agents, we have selected some approaches for the test set that deal with the incorporation of cooperative abilities into an agent framework (see Table 3).

Architecture	Reference
MAGSY	[12]
GRATE*	[18] [27]
MECCA	[39]
COSY	[7]

Table 3. Interactive agents

The three categories discussed so far suffer from different shortcomings: whereas purely reactive systems have a limited scope insofar as they can hardly implement goal-directed behavior, 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, are layered architectures. The main idea is to structure the functions of an agent into two or more hierarchically organized layers that interact with each other in order to achieve coherent behavior of the agent as a whole.

Table 4 shows the layered architectures that we consider in this paper.

Architecture	Reference
RAPs, ATLANTIS, 3T	[11] [14] [4]
Lyons & Hendriks	[22]
TouringMachines	[10]
INTERRAP	[25]
SIM_AGENT	[37]
NMRA	[30]

Table 4. Layered approaches

The last category, others, is a collection of miscellaneous agent systems, including three sub-areas, i.e., *believable agents*, *softbots*, and various Internet software agents. The instances under consideration are shown in Table 5.

Architecture	Reference
Tok	[34]
VET	[19]
ShopBot	[16]
Zuno VRISKO, QuarterDeck WebCompass, AgentSoft LifeAgent Pro, FireFly . . .	—

Table 5. Other approaches

In the remainder of this paper, we will identify and classify the core application areas of each architecture in the test set, and establish some guidelines to assist a system designer in selecting an architecture for a specific application.

3 Application Areas for Intelligent Agents

In the absence of theories to determine which agent paradigm is most useful for which class of applications, we take an empirical approach: we analyze the main areas of application known from the literature for each of the agent architectures described in Section 2. The first two columns of Table 6 in Appendix A summarize the main application areas for the architectures under consideration. While we refer to the appendix for details, in this section we discuss some interesting observations relating to Table 6.

Observation 1 Most architectures discussed are used for autonomous control systems.
The first striking observation is that a large percentage of applications (approximately half of them) 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, 3T, INTERRAP, or SIM_AGENT.

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 decentralized 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 modeling 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.

Observation 3 Distributed resource allocation is a core area for interacting agents.
The third observation is that a considerable class of applications found in Table 6 deals with distributed resource allocation, routing, and scheduling problems. Examples are logistics and transport planning, production planning systems, workflow management, and business process enactment and monitoring systems.

Observation 4 Cooperative expert systems are a core area for 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 (e.g., GRATE* as used in ARCHON, or the spacecraft health management component of the NMRA architecture).

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 modeling is a well-established discipline in AI and that many of its aspects have been revived by research areas such as *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 6 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 tend to 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 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 personalized 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 6, 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 5.

4 An Taxonomy of Agent Applications

Based on the observations made in Section 3, 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 first of which is 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 second dimension is 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, 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 agents, 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: In [45], the editors 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.

This taxonomy allows us to distinguish between six different agent types:

1. *Autonomous hardware agents (HW-AU):* They are characterized 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 6 fall into this category, e.g., RAPs, NMRA, and AuRA. An example for an agent architecture of type HW-AU presented in this volume is HEIR [31].
2. *Autonomous software agents (SW-AU):* Software systems that act autonomously and make decisions in a 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 could be seen as consisting of a front-end software assistant agent maintaining the user profile, pre-processing user input and presenting results, and of an autonomous software agent that goes shopping in the Internet. Wooldridge and Parson's abstract agent architecture presented in this volume [46] can be regarded as a further example of type SW-AU.
3. *Hardware assistant agents (HW-AS):* Hardware agents whose primary task it is to assist human users. One class of these agents are household robots. From the architectures in the test set, only 3T was used (among others) for human assistance purposes, e.g., as a wheelchair and for running errands. However, we suggest considering 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 (SW-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 (e.g., Creatures) and educational agents would fall into this category as well, according to the above definition. While there are no specific examples of architectures for assistant agents in this volume (neither HW-AS nor SW-AS), awareness of concepts such as norms [8] and moral sentiments [3] will be important for agents interacting with humans.

5. *Hardware multiagents (HW-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 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. As an example to be found in this volume, see Stone and Veloso's team member architecture [40]; also in [31], the author points out that his approach can support agents of type HW-MA.
6. *Software multiagents (SW-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. Numerous approaches found in this volume describe architectures for software multiagents. See e.g., A-Teams [32]; the agent architecture used by Skarmeas and Clark [36]; the model underlying Agentis [20], which provides an interaction model for dMARS; and PROSA$_2$ [29].

For each agent architecture considered in this paper, the third column of Table 6 shows what types of agents were built using this architecture.

5 Agent Architectures and Applications: Some Guidelines

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, algorithmic 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. Realistically, we should not hope for more than providing useful guidelines.

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 looking at agent technologies:

- 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 behavior;
- 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 decentralized 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 requester 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 Use Table 6 for a rough orientation. 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, in this case, if your problem is P, at least you have some positive evidence that A will work.

Guideline 3 If your problem requires autonomous hardware agents, then you may be well served with a hybrid architecture. 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-organizing behavior 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, has been developed at Michigan University.

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 personalization right. These, however, are much more problems of human-computer interaction (HCI), user modeling 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 of transporting 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-centered architectures presented in Section 2 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 by far too heavy-weight. If the domain you are working in is a software domain, more likely than not your architecture of choice should:

- be capable of multi-tasking;
- de-couple 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. However, to our knowledge none of the described architectures offers a uniform and complete model for adaptability.

6 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 [13] or Nwana [28]. 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 4 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 analyzing 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 6. 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., materializing 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.

7 Conclusion

The main contribution of this paper is threefold: Firstly, we analyzed a collection of agent architectures and systems built on the basis of these architectures with respect to what application areas they were used in. Secondly, we propose a taxonomy that allows us to classify agents with respect to types of applications they were used in. Thirdly, based on our analysis and the taxonomy, we extract guidelines as to which type of architecture is best suited for which type of application.

Clearly, the fuzzy nature of the task implies that there are a number of limitations, some of which were discussed in Section 6. For instance, the analysis of most approaches relies on information contained in the literature and to a degree on word of mouth information. In particular, our information relating to what applications were built using an architecture is prone to be somewhat incomplete. Nevertheless, we believe that the results described in this paper exceeds the scope of a traditional survey paper in that we aim at assisting system designers in finding the right agent to do the right thing.

References

1. P. E. Agre and D. Chapman. What are plans for? In *[24]*, pages 17–34. 1990.
2. R. C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. In *[24]*, pages 105–122. 1990.
3. A. L. C. Bazzan, R. H. Bordini, and J. A. Campbell. Moral sentiments in multi-agent systems. In this volume, pages 113–131.
4. R. P. Bonasso, D. Kortenkamp, D. P. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. In *[45]*, pages 187–202. Springer-Verlag, 1996.
5. M. E. Bratman, D. J. Israel, and M. E. Pollack. Toward an architecture for resource-bounded agents. Technical Report CSLI-87-104, Center for the Study of Language and Information, SRI and Stanford University, August 1987.
6. Rodney A. Brooks. A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, volume RA-2 (1), pages 14–23, April 1986.
7. B. Burmeister and K. Sundermeyer. Cooperative problem-solving guided by intentions and perception. In Y. Demazeau and E. Werner, eds., *Decentralized A. I.*, volume 3. North-Holland, 1992.
8. R. Conte, C. Castelfranchi, and F. Dignum. Autonomous norm-acceptance. In this volume, pages 99–112.
9. J. Ferber. Eco-problem solving: How to solve a problem by interactions. In *Proceedings of the 9th Workshop on DAI*, pages 113–128, 1989.
10. I. A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, Computer Laboratory, University of Cambridge, UK,, 1992.
11. R. James Firby. *Adaptive Execution in Dynamic Domains*. PhD thesis, Yale University, Computer Science Department, 1989. Also published as Technical Report YALEU/CSD/RR#672.
12. K. Fischer. *Verteiltes und kooperatives Planen in einer flexiblen Fertigungsumgebung*. DISKI, Dissertationen zur Künstlichen Intelligenz. infix, 1993.
13. S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, eds., *Intelligent Agents III*,

volume 1193 of Lecture Notes in Artificial Intelligence, pages 21–36. Springer-Verlag, Heidelberg, 1997.

14. E. Gat. *Reliable Goal-directed Reactive Control for Real-World Autonomous Mobile Robots*. PhD thesis, Virginia Polytechnic and State University, Blacksburg, Virginia, 1991.
15. M. P. Georgeff and F. F. Ingrand. Decision-making in embedded reasoning systems. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, pages 972–978, 1989.
16. S. Grand, D. Cliff, and A. Malhotra. Creatures: Artificial life autonomous software agents for home entertainment. In W. Lewis Johnson, editor, *Proceedings of the First International Conference on Autonomous Agents*, pages 22–29. ACM, 1997.
17. N. R. Jennings. Towards a cooperation knowledge level for collaborative problem solving. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 224–228, Vienna, 1992.
18. N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien, and M. E. Wiegand. Agent-based business process management. *International Journal of Cooperative Information Systems*, 5(2&3):105–130, 1996.
19. W. L. Johnson and J. Rickel. Integrating pedagocial capabilities in a virtual environment agent. In *Proceedings of the First International Conference on Autonomous Agents*, pages 30–38. ACM Press, 1997.
20. D. Kinny. The AGENTIS agent interaction model. In this volume, pages 331–344.
21. J. E. Laird, A. Newell, and P. S. Rosenbloom. SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–62, 1987.
22. D. M. Lyons and A. J. Hendriks. A practical approach to integrating reaction and deliberation. In *Proceedings of the 1st International Conference on AI Planning Systems (AIPS)*, pages 153–162, San Mateo, CA, June 1992. Morgan Kaufmann.
23. P. Maes. The dynamics of action selection. In *Proceedings of IJCAI-89*, pages 991–997, Detroit, Michigan, August 1989.
24. P. Maes, editor. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*. MIT/Elsevier, 1990.
25. J. P. Müller. *The Design of Autonomous Agents — A Layered Approach*, volume 1177 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Heidelberg, 1996.
26. J. P. Müller, editor. *Online Proceedings of the First International Conference on Autonomous Agents (Agents'97)*. ACM SIGART, 1997.
27. T. J. Norman, N. R. Jennings, P. Faratin, and E. H. Mamdani. Designing and implementing a multi-agent architecture for business process management. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, eds., *Intelligent Agents III*, volume 1193 of Lecture Notes in Artificial Intelligence, pages 261–276. Springer-Verlag, Heidelberg, 1997.
28. H. S. Nwana. Software agents: an overview. *Knowledge Engineering Review*, 11(3):205–244, 1996.
29. S. Ossowski and A. García-Serrano. Social structure in artificial agent societies: Implications for autonomous problem-solving agents. In this volume, pages 133–148.
30. B. Pell, D .E. Bernhard, S. A. Chien, E. Gat, N. Muscettola, P. Pandurang Nayak, M. D. Wagner, and B. C. Williams. An autonomous spacecraft agent prototype. In W. Lewis Johnson, editor, *Proceedings of the First International Conference on Autonomous Agents*, pages 253–261. ACM, 1997.
31. M. Piaggio. Heir — a non-hierarchical hybrid architecture for intelligent robots. In this volume, pages 243–259.
32. J. Rachlin, R. Goodwin, S. Murthy, R. Akkiraju, F. Wu, S. Kumaran, and R. Das. A-teams: An agent architecture for optimization and decision-support. In this volume, pages 261–276.

33. A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.
34. W. S. N. Reilly. *Believable Social and Emotional Agents*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1996.
35. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.
36. N. Skarmeas and K. L. Clark. Content based routing as the basis for intra-agent communication. In this volume, pages 345–362.
37. A. Sloman and R. Poli. SIM_AGENT: A toolkit for exploring agent designs. In *[45]*, pages 392–407. Springer-Verlag, 1996.
38. L. Steels. Cooperation between distributed agents through self-organization. In Y. Demazeau and J.-P. Müller, eds., *Decentralized A.I.*, pages 175–196. North-Holland, 1990.
39. D. D. Steiner, A. Burt, M. Kolb, and Ch. Lerin. The conceptual framework of MAI^{2}L. In *Pre-Proceedings of MAAMAW'93*, Neuchâtel, Switzerland, August 1993.
40. P. Stone and M. Veloso. Task decomposition and dynamic role assignment for real-time strategic teamwork. In this volume, pages 293–308.
41. M. Tambe, R. Jones, J. E. Laird, P. S. Rosenbloom, and K. B. Schwamb. Building believable agents for simulation environments. In *Proceedings of the AAAI Spring Symposium: Believable Agents*. AAAI, 1994.
42. S. R. Thomas. The PLACA agent programming language. In *[44]*, pages 355–370. 1995.
43. D. E. Wilkins, K. L. Myers, and L. P. Wesley. Cypress: Planning and reacting under uncertainty. In M. H. Burstein, editor, *ARPA/Rome Laboratory Planning and Scheduling Initiative Workshop Proceedings*, pages 111–120. Morgan Kaufmann Publishers Inc., San Mateo, CA, 1994.
44. M. J. Wooldridge and N. R. Jennings, editors. *Intelligent Agents – Theories, Architectures, and Languages*, volume 890 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1995.
45. M. J. Wooldridge, J. P. Müller, and M. Tambe, editors. *Intelligent Agents II*, volume 1037 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1996.
46. M. J. Wooldridge and S. D. Parsons. Intention reconsideration reconsidered. In this volume, pages 63–79.

A Agent architectures, types, and applications

Table 6 overviews the test set used in this paper (first column), the main types of applications that were built using these architectures (second column), and the corresponding classification of the types of agents built using these architectures according to Section 4 (third column). For each architecture mentioned, at least one reference is given in the fourth column of the table.

Architecture	Applications	Classification	Reference
Reactive agents			
Subsumption architecture	mobile robots and land vehicles	HW-AU	[6]
Self-organizing agents	mobile robots, emerging group behavior	HW-AU, (HW-MAS)	[38]
AuRA	mobile robots and land vehicles	HW-AU, (HW-MAS)	[2]
Dynamic action selection	technique has been used in mobile robots and Artificial Life applications	HW-AU	[23]
PENGI	arcade computer game	SW-AU	[1]
ECO model	distributed problem-solving	SW-MA	[9]
Deliberative agents			
IRMA	general architecture, probably robotics	HW-AU	[5]
PRS	originally: robotics	HW-AU	[15]
dMARS	air traffic control, business process enactment and monitoring	HW-AU, SW-AU, SW-AS, (SW-MA)	[33]
SOAR	general AI problem-solving architecture; mainly autonomous control systems, more recently also believable agents	HW-AU, SW-AU , SW-AS	[21] [41]
Cypress	mobile robots and vehicles	HW-AU	[43]
Agent0 / PLACA	no specific application mentioned, supports communicative acts	HW-MA (?), SW-MA (?)	[35] [42]
Interacting agents			
MAGSY	production planning, distributed resource allocation, cooperating expert systems	SW-MA, SW-AS	[12]
GRATE*	electricity network diagnosis, later: workflow management	SW-MA, SW-AS	[17] [18] [27]
MECCA	traffic control systems, personal digital assistants	SW-AS, SW-MA	[39]
COSY	production planning, transport planning	SW-MA	[7]
Layered approaches			
RAPs, ATLANTIS, 3T	mobile robots and land vehicles, robotics wheelchair, errand running (3T)	HW-AU, (HW-AS)	[11] [14], [4]
Lyons & Hendriks	mobile robots and land vehicles	HW-AU	[22]
TouringMachines	mobile robots and land vehicles	HW-AU, (SW-AU)	[10]
INTERRAP	cooperating robots, flexible transport systems, game playing agents, logistics	HW-MA, (SW-MA)	[25]
SimAgent	humanoid robots or softbots (very abstract architecture)	HW-AU (?), SW-AU (?)	[37]
NMRA	spacecraft control	HW-AU	[30]
Other approaches			
Tok	believable agents for entertainment and arts applications.	SW-AS	[34]
VET	education and training	SW-AS	[19]
ShopBot	Resource discovery, electronic commerce	SW-AS, SW-AU	[16]
Zuno VRISKO, QuarterDeck WebCompass, AgentSoft LifeAgent Pro, FireFly . . .	information retrieval and information filtering, personalization	SW-AU, SW-AS SW-AU, SW-AS	—

Table 6. Agent architectures and agent applications