SIM_AGENT two years on

Jeremy Baxter, **Richard Hepplewhite**,
Defence Evaluation and Research Agency
Malvern, Worcestershire WR14 3PS

Brian Logan and **Aaron Sloman**
School of Computer Science
University of Birmingham, Birmingham B15 2TT

Abstract

At ATAL'95 a paper was presented reporting on the SIM_AGENT toolkit [8]. SIM_AGENT was developed to provide a flexible framework for the exploration of architectures for autonomous agents consisting of a variety of concurrent interacting modules operating in discrete time. The previous paper outlined two early experiments with the toolkit. In this paper, we describe the experiences of two groups actively using the toolkit and report some of what we have learnt about its strengths and weaknesses. We briefly describe how the toolkit has developed since 1995 and sketch some of the ways in which it might be improved.

1 Introduction

At ATAL'95 a paper was presented reporting on the SIM_AGENT toolkit [8]. SIM_AGENT was developed to provide a flexible framework within which to explore diverse agent architectures, including agents containing a variety of concurrent interacting modules of various types, such as modules concerned with perception, motive generation, communication, planning, learning, self-monitoring, and plan execution. It was expected that some agents could include a mixture of reactive and deliberative mechanisms, and in addition some might need 'meta-management' mechanisms involving self monitoring, self evaluation and self modification. We also expected some of the mechanisms to be symbolic and others sub-symbolic with convenient interfaces between them (e.g. between condition-action rules and a neural net).

The previous paper outlined two early experiments with the toolkit: a simple simulated wall-following robot with a reconfigurable mixture of neural nets and symbolic rules; and the co-evolution of two agents developing a language for collaborative problem solving. Since then there have been a number of different users of the toolkit at Birmingham, including Ian Wright, who has experimented with a complex architecture for a motivated agent with meta-management capabilities [11], Darryl Davis who reported experiments with cooperative agents with different levels of intelligence at ATAL'96 [1], and Riccardo Poli who has continued to use the toolkit for evolutionary experiments, with his students. The main users, however, are Brian Logan who is working with Aaron Sloman at Birmingham exploring multi-level architectures for human-like agents [10] and Jeremy Baxter and Richard Hepplewhite at DERA Malvern who

are developing simulated battle scenarios which might be useful for training novice tank commanders [4].

In this paper we present some of what we have learnt about the strengths and weaknesses of the toolkit based on our experience to date, and sketch some of the ways in which it might be improved. We see this as part of a continuing process through which developers and users of different toolkits can share ideas and learn about each others goals, problems, solutions and unanswered questions, so that the community as a whole can develop a better understanding of the issues. In the next section, we briefly describe the SIM_AGENT toolkit. Section 3 reports on some of the experiences of the groups at Birmingham and the DERA, including problems which led to modifications of the toolkit. We then outline some of the changes in the toolkit since 1995, and go on to discuss some of the additional changes that now seem desirable, and long term prospects.

2 The SIM_AGENT toolkit

In this section, we briefly describe the original version of the toolkit, as reported in [8]. A more detailed description of the toolkit can be found in [9].

SIM_AGENT was developed to support the exploration of agent architectures for one or more agents operating in discrete time. It arose out of a long term research project concerned with architectures for autonomous agents with human like capabilities including multiple asynchronous sources of motivation and the ability to reflect on which motives to adopt, when to achieve them, how to achieve them, how to interleave plans and so on. The toolkit is based on the POPRULEBASE library developed at the University of Birmingham. POPRULEBASE is a flexible forward chaining production system interpreter for 'rule-based' programming.

An agent consists of a number of interacting components or mechanisms, some of which may be resource limited. Each internal mechanism is represented primarily by a set of condition-action rules, where the conditions are either simple patterns that can be matched against entries in the database for that agent, or else more complex conditions, including disjunctions, negated conditions, implications, existentially quantified conditions and 'WHERE' conditions which can invoke arbitrary Pop-11 code producing a boolean result. When a rule has all its conditions satisfied its actions can all be run. Actions may be simple lists to be added to or deleted from the agent's database, or more complex actions including 'POP11' actions invoking arbitrary Pop-11 instructions. Rules are grouped into rulesets. In general, rulesets (along with the Pop-11 code they invoke) correspond to or implement the different components or mechanisms of an agent architecture, for example perceptual processing, reactive behaviours such as obstacle avoidance, deliberative behaviours such as planning, motive generation etc.

In each timeslice every agent's internal mechanisms get a chance to run. For each agent in turn, the scheduler runs each of the agent's rulesets by calling the POPRULEBASE interpreter. The rule interpreter is given a ruleset and a database, and it repeatedly runs the rules in the ruleset until either some specified cycle limit has been reached, or a 'STOP' action is executed. By default the order of the rules in a ruleset determines the order in which their conditions should be tested, and it may also determine which rules run if several have all their conditions satisfied. However, the designer can over-ride the default rule selection strategy for particular sub-mechanisms. Options include (a) running all the rules, (b) running only the first runnable rule, (c) creating a set of rule instances and allowing a user procedure to order or filter them.

The toolkit is implemented in Pop-11 and is intended to be very 'open' with all the source code available to users and easy interfaces between the toolkit mechanisms and Pop-11 code, so that users can extend or modify the toolkit's functionality as needed, e.g. by mixing Pop-11 code with the rule-based notation of the toolkit.

3 Experience with the toolkit

Our assumption was that different application domains would require different sorts of agents, and even as regards the goal of modelling human cognitive processes it was still premature to adopt any particular agent architecture, since current theories were bound to be oversimplified. The toolkit was intended to provide an environment for exploring alternative architectures, which would provide useful experience helping us to test theories and to learn about the strengths and weaknesses of alternative designs. In the process we would also learn more about the toolkit and how it can be improved. In this section, we present a preliminary analysis of the strengths and weaknesses of the toolkit, based on our experience so far.

3.1 Operating in real time

The toolkit is being used at DERA Malvern [4] to simulate commanders within a real-time battlefield simulation. The agents interact with an external physical simulation of the battlefield environment. In order to tie in with the simulation it was a reasonably straightforward task to use communication from and to the simulation to provide the sensors and effectors for agents.

Other entities within the external simulation may be controlled by humans, and it is therefore necessary for the SIM_AGENT agents to run in real time. The toolkit was intended to support architectures with resource limits and provides mechanisms for varying relative speeds of components within an architecture as well as relative processing speeds of different agents. However, the emphasis was on simulated time rather than real time, since experimental implementations of particular components (e.g. neural nets) might be very slow compared with later implementations using special purpose hardware or multiple processors. As a result, none of the built-in mechanisms of the toolkit include any reference to real time or the processor time, only to counts of rule firings.

The only way of obtaining real-time response was to use the the 'cycle limits' for each ruleset to control the number of rules that can fire in a single cycle (single pass of the scheduler) thereby constraining the amount amount of processing an agent could perform, while updating the agents from the external simulation at regular time steps (typically two seconds). Each agent runs up to its cycle limit (or completion) in the first pass of the scheduler at which point the scheduler checks to see if the next time step has been reached. If more time is available another pass is made through all active agents. This continues until all agents have completed (have no more rules they can run) or the time step is reached. It is therefore impossible for an agent to predict how much processor time it will get in any slice of real time since this depends on the number of agents running and how much work they are doing. This requires the use of 'anytime' techniques which always have a solution of some sort available so that the agent can start executing it when desired.

Although not explicitly designed with real-time performance in mind, the toolkit works surprisingly well in this application. However, the simple approach outlined

above results in a number of inefficiencies in that all agents run, even if they have completed processing, and all rules in all rulesets are examined even if they can do no work until the next timestep (when new information might trigger them). In order to ensure real-time response, it is important to ensure that the actions performed by rules take a small amount of time. This typically means that the rules must be relatively 'fine grained' which in turn implies that we need a lot of rules, each of which must match its conditions and store its results (i.e. the current state of the problem solving process) in the database. Because there is such a variety of types of conditions and actions, including user defined condition types and action types, no standard indexing scheme can be used to optimise the invocation of rules.[1] It may be possible to do some partial indexing where simple rule and action forms are used, though this will always be only a partial solution, of limited value when the full flexibility of POPRULEBASE is exploited. As a result the system can be very wasteful, repeatedly checking the same conditions to find out which rules are runnable.

3.2 Maintaining dependencies

In work at Birmingham, we are exploring architectures for agents which play a variant of the game of 'hide-and-seek' in complex terrains. Our agents operate in a dynamic environment on the basis of incomplete or uncertain information, revising their beliefs and intentions if the information on which they are based changes or the assumptions made by the agent turn out to be incorrect. It is often the case that there is a delay between an agent forming an intention and acting upon it. Moreover, an intention may take some time to achieve (e.g. it may require the execution of a multi-step plan). This involves keeping track of dependencies between the agent's beliefs and intentions and between its intentions and the plans to achieve these intentions. In addition, where the information available to an agent is known to be incomplete or uncertain, it may wish to do some simple hypothetical reasoning, for example to do some contingency planning, or to investigate the implications of differing assumptions.

Maintaining dependencies between an agent's beliefs and intentions has been a recurring theme of the work at Birmingham, and several alternative strategies have been employed, including 'guard conditions' (or procedures) on rule and action execution, additional 'consistency checking' rulesets which determine the validity of the current intention, and 'context tags' or separate databases for hypothetical reasoning. However none of these approaches is entirely satisfactory. It can be difficult to anticipate the guard conditions if the agent has, for example, a general planning capability. Moreover such conditions tend to proliferate and are hard to maintain as the agent's capabilities develop. Using 'context tags' or separate databases for hypothetical reasoning means that it is difficult to share information between contexts, which can result in the rederivation of the same information in different contexts.

3.3 Defining agent capabilities

The fact that the toolkit is *embedded* in a general purpose programming language (in this case the Poplog programming environment), rather than *implemented* in a programming language (as, say, an interpreter for a logical agent language might be) gives us considerable flexibility in defining the agent's primitive actions or capabilities. There

[1] For example, it means we can't use the RETE algorithm to optimise rule selection.

are often significant advantages to using an appropriate level of abstraction, and it is not always clear what this should be when developing an architecture.

In some cases, the designer's initial choice of primitive actions turns out to be inappropriate. For example, the original design for our hide-and-seek agents incorporated a simple route planning capability as a primitive action. The planner was implemented in Pop-11, on the assumption that an encapsulated mechanism would simplify the integration of planning with other behaviours. In practice, this turned out to be too inflexible, and in more recent work we have split the planner into a number of more basic primitive planning actions which perform model abstraction, step planning, plan refinement and smoothing, controlled by a collection of meta-management rules, which decide when to plan, what sorts of plans are required and how much effort the agent can afford to spend on planning.

While it would have been possible to (re)implement the planner directly in POPRULEBASE, this would have required considerable effort and entailed a significant computational overhead. The ease with which code fragments can be embedded in the conditions and actions of rules means that those parts of the problem which are not part of the general architecture can be implemented procedurally, with a consequent increase in both clarity and efficiency.

4 How the toolkit has developed

A number of developments of the toolkit occurred after it began to be used in earnest. Some of these were anticipated at the time the toolkit was developed, for example, the addition of tools for profiling. Other changes are more interesting, as they were concerned with the expressive power of the toolkit and the control mechanisms, i.e. they made a difference to how easy it was for the designer to achieve desired effects. In this section, we briefly describe some of the more recent extensions of the toolkit.

4.1 Controlling attention

One of the main problems with the toolkit was the inefficiency of POPRULEBASE. Each agent typically contains a large number of rules which must be matched against the agent's database. Although the rules are organised into rulesets, by default all of each agent's rulesets are run at each timeslice, so all the rules are considered. Because of the difficulty of indexing the rules, our approach has to been to make it easier to control which rulesets are considered by the interpreter. This was achieved by introducing the *rulefamily* as a type of data-structure which could maintain the context associated with a collection of rules.

A rulefamily is a collection of cooperating rulesets. A particular sub-module in the architecture of an agent can implemented as a ruleset or rulefamily operating either on a general database in the agent or on a collection of specialised databases within the agent. Only one ruleset in the family is ever in control at a time, but the current ruleset can transfer control to another ruleset in the family using various actions provided for this purpose, including actions for pushing and popping rulesets. In addition, it is possible to push and pop databases so as to restrict the database checking required in a particular context. This allows a complex rule-based system to be divided into a set of different rulesets, such that only one relatively small ruleset is active at any one time. The division of internal agent mechanisms into rulefamilies has the obvious merit of supporting modular design with easy addition and removal or invocation of particular

rulesets, but it can also reduce the amount of wasteful condition-checking, since in different contexts an agent can enable and disable different rulesets and databases.

In order to efficiently control the agents running in real time it was also necessary to modify the scheduler to maintain information about the number of rule firings which took place and detect when an agent had finished processing. This was combined with a new facility to control the list of agents being run on each cycle. Each time the scheduler is run it is given a list of agents to execute, and each agent reports if it did not fire any rules on a pass, allowing it to be removed from the list of runnable agents until new information arrives.[2]

4.2 Dynamic local control environments

Some of these mechanisms were available in previous versions of the toolkit but they were hard to use, and they also introduced bugs because the rule interpreter was originally designed to run as a single process and did not save the context properly when switching from one agent to another or from one sub-mechanism within an agent to another. However when running a variety of different sorts of concurrent mechanisms in different agents, it is necessary to be able to save and restore the variable value environments required by those different mechanisms. For example, one of the problems of the original implementation was that the mechanism for selection of rules and sorting or filtering of rule-instances was controlled by global variables, making it difficult to use different selection strategies for different rulesets.

Pop-11 already has a '`dlocal`' mechanism which generalises dynamic variable binding and can be used to ensure that whenever a procedure exits whether normally or abnormally some 'exit action' is run. By allowing 'DLOCAL expressions' to be associated with individual rules, rulesets, rulefamilies or complete rulesystems, we allowed control environments to be safely modified by individual mechanisms without affecting anything else, as the pre-existing environment is automatically restored when the local control environment established by the DLOCAL expression is exited.

A side-effect of the introduction of DLOCAL expressions was more effective debugging tools. In a simulation involving multiple concurrently acting agents, each of which could have multiple concurrent internal mechanisms, the task of keeping track of what was going on could be extremely difficult, with megabytes of trace printing generated if all the system trace procedures are turned on. Using DLOCAL expressions it became possible to require a particular suspect rule or ruleset to turn on debug tracing without having it turned on more globally, requiring far less sifting through irrelevant printout.

However, the most important change to help with this problem of understanding what was happening was the addition of a graphical package integrated with the objectclass system and based on the the Pop-11 'relative coordinates' graphical package.[3] This made it easy for changes occurring in simulated agents to be automatically displayed in a screen (e.g. changes of location and orientation, but other changes also). Moreover, since the new mechanisms allowed mouse controlled movements of picture objects to produce changes in the corresponding simulated agents, it became much easier to test simulations and demonstrate that they were working. The graphical package

[2] This strategy works in applications where no rules fire after quiescence unless there are new inputs (percepts or messages), i.e. where the user has not implemented counters or timers as 'WHERE' rule conditions.

[3] Prior to the introduction of these general graphical tools the DERA team were able to develop their own graphical displays by invoking the basic Pop-11 graphical facilities from the toolkit, another example of the advantages of having the toolkit embedded in a general purpose programming language.

is still being extended to provide a range of asynchronous control mechanisms, e.g. enabling the states of individual objects to be interrogated easily at run time.

4.3 Belief and intention revision

While it is possible to use the basic facilities of the toolkit to keep track of dependencies between the agent's beliefs and intentions, it requires considerable effort on the part of the designer to ensure that the necessary bookkeeping information is generated by the various modules and that this information is used in an appropriate way. There is also often a considerable overhead in checking the consistency conditions in rules.

In an attempt to overcome these problems, we are experimenting with moving dependency maintenance out of the agent and into the toolkit itself. Maintaining dependencies is a recurring problem in artificial intelligence. A standard approach to this problem is a 'truth maintenance system'. A truth maintenance system is a system for reasoning about dependencies between pieces of information. There are many different types of truth maintenance system with varying capabilities, but at their simplest they perform 'bookkeeping' for a problem-solver, accepting information from the problem-solver about dependencies and inconsistencies and informing the problem solver about which data can consistently be used as a basis for deliberation. We have implemented a simple assumption-based truth maintenance system (ATMS) [2] in Pop-11 as an extension to the toolkit.[4] Moving the bookkeeping associated with dependency maintenance out of the agent and into the toolkit allows us to tackle problems with more complex dependencies and, more importantly, to focus on the more interesting problem of *which* dependencies are important.

4.4 Multi-agent systems

Even with the efficiency improvements outlined above, in many cases we required larger numbers of agents than could be reasonably simulated on one machine, and it became necessary to modify the toolkit so that several copies could be run simultaneously and communicate with each other. In order to allow the toolkit to be distributed two main features were required: a common setup procedure which allows distributed SIM_AGENT's to start simultaneously with shared data; and a system for passing messages between agents on different machines. Simultaneous start up is relatively simple with a common setup file and broadcast sockets being used to synchronise the internal clocks and assign unique names to agents. The passing of messages was a little more complex but, because of the modular nature of the toolkit, could be achieved by method overloading in a manner which was transparent to the internal mechanisms of the agents. The toolkit's message passing method which took lists from the database of the transmitting agent and placed it in the incoming message slot of the recipient was overloaded with one which broadcast the message on a UDP broadcast socket.

5 Further work

In this section we comment on some problems with the developments of the toolkit reported in the previous section, and discuss some of the additional changes to the toolkit that now seem desirable in the light of our experience.

[4] This is similar to the approach to single agent belief revision adopted by Malheiro and Oliveira [5].

The rulefamily mechanism provides an extra level of abstraction which simplifies the implementation of certain control strategies. As an efficiency measure rulefamilies have been only partially successful but they introduced some new concepts which could be used to control how an agent focused its attention by allowing some internal state to be maintained, most importantly which of a family of rulesets is currently active. For example, it made possible 'setup rulesets' which ran only once and then replaced themselves with active versions. It also allows rulesets to be focused on a particular aspect of a problem over several cycles without the need to continually check their applicability. When a ruleset has finished it transfers control to another ruleset or switches back to a 'control ruleset' which decides what to do next. For example, this allows a straightforward implementation of the idea of a 'problem space', though in this case, the user is responsible for programming the transitions between problem spaces.[5]

This is rather like using gotos to control the flow of program execution. Apart from the worry that such systems will be difficult to modify and maintain (which is largely speculation), there are problems with how rulefamilies interact with other features of POPRULEBASE. In particular, there are problems if all the rules in a ruleset are executed in parallel.[6] In this case, all the actions conceptually happen at once, and it is bad style to rely on any specific execution order for the rule instances. For example, if one of the rule instances contains a SWITCHRULESET action, do the other rule instances get run or not? In practice, only some of them will, depending on the position of the rule containing the SWITCHRULESET in the current ruleset. This is not a requirement that all combinations of the toolkit features be 'consistent', but it is a significant problem that one of the mechanisms provided for meta-level control is incompatible with the parallel execution of rules.

Some users have addressed this by allowing the use of condition-action rules where the actions include Pop-11 code to run the rule interpreter on a specified ruleset. However the recursive invocation of the interpreter is not integrated with the resource control mechanisms provided in SIM_AGENT, which can be a problem if the ruleset does not run to completion within a single timeslice. A better approach would be to provide a special form of action to invoke the rule interpreter with a specified ruleset (and maybe a cycle limit to control their resource consumption as already happens for the rulesets run by the scheduler). The sub-ruleset could share the same database as the invoking rule, or the rule could create a private database for the sub-ruleset to do its work in. This would provide more flexibility than the rulefamily mechanism which assumes a fixed set of rule-families is built into each agent's rulesystem, making dynamic introduction of new rulesets very awkward (though not impossible). The new proposal would allow a hierarchy of levels of control, with a rule at one level deciding to run a ruleset for a given number of cycles or until quiescence, possibly interleaved with execution of the rules at the 'control level' monitoring the progress of the sub-ruleset and deciding when it is appropriate to terminate it.[7] It could also simplify the modelling of one agent by another.

Such an approach addresses the three motivations for introduction of rulefamilies, namely efficiency, modularity, and cognitive modelling, and it has the additional advantage that the control knowledge is kept out of the low level rulesets. It also allows

[5] In contrast to say, SOAR, where the problem spaces are generated automatically by the architecture [7].

[6] We have found the parallel execution of rulesets to be very useful in certain situations, particularly in the implementation of low-level behaviours such as perceptual processing, for example when there are multiple interpretations of the percepts.

[7] For example, it would allow the implementation of the metalevel control of PRS [3].

(simulated) concurrent control, at the cost of introducing some concepts/syntax to cope with the notion of a sub task or process started by a 'control' rule, and ways of controlling that task.

Another disadvantage of storing some state in rulefamilies arose from the need to allow agents to change their mechanisms, e.g. through learning or 'promotion' to a new role. Changing the role of an agent by giving it a different rulesystem became harder to do: instead of simply gaining a different list of rules and continuing to run when an agent changes role (for instance from an individual tank commander to the commander of a group) the state of any retained rulefamilies must be preserved. This involves copying the internal state of the agent's currently active rulefamilies and initialising any new rulefamilies it gains to their default state.

We anticipate further extensions to the toolkit will be necessary in the future. Some, like the ATMS, will be general purpose, while others will be domain-specific. For example, it may be necessary to extend the toolkit's representations of its own processes to support the meta-management capabilities implied by the recursive invocation of the rule interpreter. The ease with which the toolkit can be extended is one important way in which it supports the development of agent-based systems and the exploration of new agent architectures. At a higher level, we are interested in abstracting away from the SIM_AGENT toolkit, with the aim of specifying high-level libraries of architectural components which could, in principle, be implemented in other toolkits. As a simple example, it would be useful to have some libraries which configure POPRULEBASE to run in the two or three most useful consistent combinations of control options.

One of the most interesting challenges is to explore ways of implementing self-knowledge, self-evaluation and self-modification. Simple forms are available in mechanisms with feedback and reinforcement learning, but those essentially only alter weights and connections between fixed structures. Human learning seems to involve more complex types of self modification, for instance learning a new strategy for acting or thinking. One of the important challenges for future research is to find good ways to enable an agent to discover the need and opportunity for such changes.

6 Final Remarks

Much work on agents and multi-agent systems is concerned with ensuring rationality of agents and provable correctness of designs or implementations. However, neither concern is essential for the scientific study of agents, partly because accurate modelling of humans and other animals with all their imperfections obviously cannot presuppose perfect rationality or perfect reliability for any particular task, and partly because even within engineering frameworks it has long been understood that there are domains in which tractability entails a rejection of perfect rationality [6].

We conjecture that experience will show that toolkits and languages that are not fettered by constraints of rationality or *provable* correctness provide a more general framework for exploring a broad range of types of agents with bounded rationality. Of course, by providing more freedom such tools also challenge the designer to take extra care in clarifying objectives, and to develop criteria and tools for evaluating systems.

We do not claim that SIM_AGENT is the only or the best such exploratory toolkit. However, at present our understanding of the space of possible designs is still so limited that dogmatic commitment to any one methodology, language or toolkit could seriously restrict future research. We would therefore encourage developers and users of different toolkits to compare each others systems and to communicate regularly about

their goals, problems, solutions and unanswered questions so that the community as a whole can develop a better understanding of the issues. On that basis a new generation of more powerful and general tools and languages may be evolved. This bootstrapping process may have to continue for several generations before we are able to model anything close to human intelligence.

In particular we welcome attempts to reimplement and improve on our tools (e.g. reimplementation using other programming languages), challenges that the toolkit may be unable to meet, and constructive suggestions regarding future developments.

Acknowledgements

The authors wish to thank the members of the Cognition and Affect and EEBIC (Evolutionary and Emergent Behaviour Intelligence and Computation) groups at the School of Computer Science, University of Birmingham for useful discussions and comments. This research is partially supported by a grant from the Defence Evaluation and Research Agency (DERA Malvern).

References

[1] D. N. Davis. Reactive and motivational agents: towards a collective minder, in *Intelligent Agents III, Agent Theories, Architectures and Languages*, J. Müller, M. Wooldridge and N. R. Jennings (Eds.), Lecture Notes in Artificial Intelligence, Springer-Verlag, pp 309–323, 1997.

[2] J. de Kleer. An assumption-based truth maintenance system, *Artificial Intelligence*, Vol. 28, pp 163–196, 1986.

[3] M. P. Georgeff and F. F. Ingrand. Decision making in embedded reasoning systems, in *Proceedings of the 6th International Join Conference on Artificial Intelligence*, pp 972–978, 1989.

[4] R. T. Hepplewhite and j. W. Baxter. Broad agents for intelligent battlefield simulation, in *Proceedings of the 6th Conference on Computer Generated Forces and Behavioural Representation*, Orlando, Florida 1996.

[5] B. Malheiro and E. Oliveira. Consistency and context management in a multi-agent belief revision testbed, in *Intelligent Agents II, Agent Theories, Architectures and Languages*, M. Wooldridge, J. Müller and M. Tambe (Eds.), Lecture Notes in Artificial Intelligence 1037, Springer-Verlag, pp 361–391, 1996.

[6] H. A. Simon. *The Sciences of the Artificial*, MIT Press, 1970.

[7] P. S. Rosenbloom, J. E. Laird and A. Newell. *The Soar Papers: Research on Integrated Intelligence*, MIT Press, 1993.

[8] A. Sloman and R. Poli. SIM_AGENT: A toolkit for exploring agent designs, in *Intelligent Agents II, Agent Theories, Architectures and Languages*, M. Wooldridge, J. Müller and M. Tambe (Eds.), Lecture Notes in Artificial Intelligence 1037, Springer-Verlag, pp 392–407, 1996.

[9] A. Sloman. *Overview of the* SIM_AGENT *toolkit*, University of Birmingham, School of Computer Science, Available at URL http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html, 1996.

[10] A. Sloman. What sort of a control system is able to have a personality, in *Creating Personalities for Synthetic Actors: Towards Autonomous Personality Agents*, R.Trappl and P. Petta (Eds.), Lecture Notes in Artificial Intelligence, Springer-Verlag, pp 166–208, 1997.

[11] I. P. Wright and A. Sloman. *Minder1: An implementation of a protoemotional agent architecture*, The University of Birmingham, School of Computer Science, Technical Report CSRP-97-1, 1997.