In *Proceedings AAAI-98 Workshop on Software Tools for Developing Agents*, Eds. Brian Logan and Jeremy Baxter, Madison, July 1998.

What's an AI Toolkit For?

Aaron Sloman
School of Computer Science
The University of Birmingham,
Birmingham B15 2TT,
England

A.Sloman@cs.bham.ac.uk,
http://www.cs.bham.ac.uk/~axs

Abstract

This paper identifies a collection of high level questions which need to be posed by designers of toolkits for developing intelligent agents (e.g. What kinds of scenarios are to be developed? What sorts of agent architectures are required? What are the scenarios to be used for? Are speed and ease of development more or less important than speed and robustness of the final system?). It then considers some of the toolkit design options relevant to these issues, including some concerned with multi-agent systems and some concerned with individual intelligent agents of high internal complexity, including human-like agents. A conflict is identified between requirements for exploring new types of agent designs and requirements for formal specification, verifiability and efficiency. The paper ends with some challenges for computer science theorists posed by complex systems of interacting agents.

Introduction

This paper is a collection of thoughts about toolkits for developing intelligent systems involving one or more agents. The top level question driving this enquiry is whether it makes sense to ask whether it is possible to produce *one* toolkit which meets all requirements, and if not why not.

It is obvious and not very interesting that we can form the union of all toolkits ever developed plus a guide to selecting the best one for each type of agent development task. This is not an interesting option because a union of that kind will just be a highly redundant mish-mash of mostly unrelated components, whereas an interesting toolkit should have greater integration, including a principled set of facilities designed to meet the needs of various kinds of researchers and developers concerned with intelligent agents.

Can something more principled be done? My hunch is that the range of scientific and engineering objectives to be found in AI is too large and diverse for any very coherent "finished" toolkit to meet them all, even if we factor out personal preferences of developers for different programming languages or styles of development.

It may be possible to produce an extendable toolkit supporting a very wide range of paradigms, though it may be complex and fairly messy. Likewise, knowledge of a human language can be thought of as possession of some sort of toolkit for communication, which is rich in generative power, and highly extendable. The linguistic toolkit includes all sorts of diverse components, including morphological, syntactic, semantic, pragmatic, mechanisms and mechanisms for extending the language. Moreover, linguistic knowledge cannot be cleanly separated from knowledge about the world in which it is used. Similarly with toolkits for developing intelligent agents.

Just as different sub-languages with varying degrees of precision and formality are appropriate for different communicative contexts, so also will different subsets of development tools be relevant to different classes of design problems.

This paper attempts to identify some of the distinctions to be made within a generic set of tools and tasks.

Of course, to the extent that there is real unclarity, or disagreement, about what constitutes AI, or what an agent is, this may be a hopelessly ill defined goal. However, for now I am going to assume that at the very least the tasks include *both* engineering goals such as producing intelligent robots, software systems, and symbiotic human-machine systems *and* scientific goals such as understanding and modelling existing intelligent systems and also trying to understand the space of possible designs, natural and artificial (Davis 1996; Sloman 1994; 1996).

I do not expect that a paper as short as this can exhaust the topic, but I'll offer some groundwork by discussing a set of questions relevant to designing or choosing a toolkit. It may be that such a survey is useful if only because some researchers, including both toolkit designers and toolkit users with limited experience, may be unaware of the full range of objectives, requirements and potentially useful tools that may be needed.

To some extent this paper was inspired by Brian Logan's paper in this workshop (Logan 1998). His paper is concerned with classifying types of agent systems, whereas I am more concerned with classifying the issues that arise in developing agent systems, though obviously the two are closely related.

The development issues include questions like: What sorts of things need to be put together? How many different ways are there of putting things together, and what are the reasons for choosing them?

Some of the answers will obviously depend on *what* is being assembled. For instance, it will make a difference whether the task is to assemble *one* complex goal-driven agent or a *collection* involving many different kinds of agents which do not necessarily share any common goal. The kind of internal complexity required within each agent will also affect the design process.

Less obviously, there are other determining factors, such as how well specified the task is initially, whether further development work may be required once the system is up and running, and what sorts of testing will be required. These and other points are discussed in more detail below.

Some questions

In considering how to design a toolkit there are a number of issues to be addressed, including the following:
(a)What kinds of scenarios are to be developed? Answering this includes specifying which types of agents and objects are involved in the scenarios, the kinds of goals, knowledge and skills they can have, and the ways they can develop and behave.
(b) What are the scenarios to be used for? For instance, is it a research activity or is there a practical goal?
(c) Is the system being developed a simulation of something else, or is it "the real thing"? (E.g. an actual plant control system.)
(d) To what extent are the objectives and design strategies already well understood in advance of use of the toolkit?
(e) Which is more important: ease and speed of development and debugging, or speed of execution of the developed system?
(f) Will thorough testing and collection of performance statistics be required?
(g) Is it essential that the developed system be provably correct as in some safety critical applications, or is it enough that it works well in a wide range of test situations?

Some of the questions break down into a number of more specific sub-questions, as will be shown below.

What kinds of scenarios are to be developed?

This is in part a question about the ontology of the system being developed, including which sorts of agents are involved, a question discussed in more detail later. Figure 1 very loosely depicts a class of scenarios where there are various kinds of concurrently active entities, including agents which can communicate with one

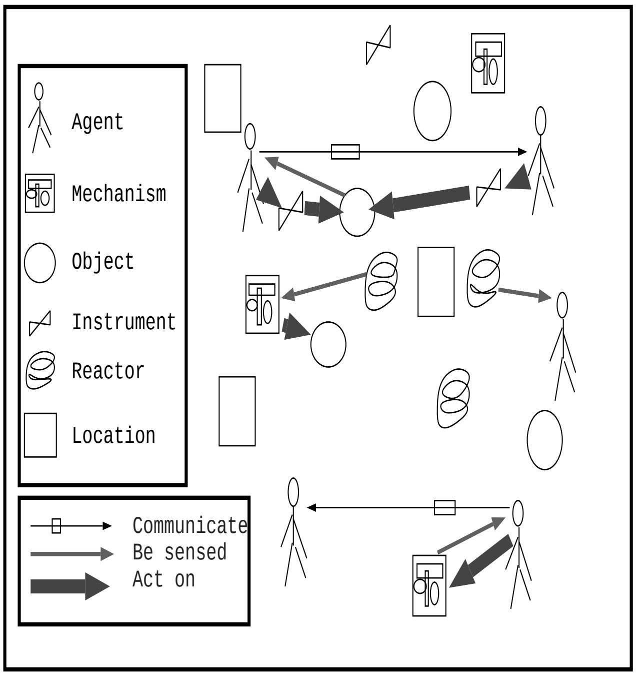

Figure 1: **Ontologies**
(In general a toolkit may be used to build a simulation involving many types of entities interacting in many different ways. A few types are indicated here.)

another, agents and objects which can sense and react to other things, instruments which can do something if controlled by an agent, "reactors" which don't do anything of their own accord but can react if acted on (e.g. a coiled spring, or mouse-trap) and immobile locations which may have variable extents and all sorts of properties relevant to agents, including, for instance continuously varying heights and other features. Such terrain features can have causal consequences, e.g. which objects are visible from a particular location, which locations are too steep to traverse, the energy resources required for travel along a particular path, etc.

This is not meant to be a complete or systematic overview, merely an indication of the diversity of participants and types of causal interactions that can occur in scenarios of interest to AI.

A toolkit to support such diversity cannot be expected to anticipate all the types of entities, causal and non-causal relationships, states, processes, etc. which can occur. So it will be desirable for users to be able to extend the ontology as needed. One approach is the use of object oriented programming, especially where multiple-inheritance is supported. A different approach is to use axioms defining different classes and subclasses. Which is more useful is likely to depend on other factors than the nature of the ontology, some discussed below.

Another important difference between scenarios concerns whether they are entirely concerned with software processes within one or more computers (along with displays and interaction devices) or whether there will also be physical objects with additional sensor and motors, e.g. robots or components of a factory being controlled by the system under development.

Tools for developing systems of the latter kind will require access to the specialised equipment including robots, machinery, sensors, etc. Requirements will include obtaining sample data, testing, and of course final deployment, though for some development work simulations may suffice, as is happening increasingly in real engineering design and development.

There are also important differences between scenarios in which artificial agents interact with humans and those in which they interact merely with one another and with hardware and software artefacts, machines, etc. Interactions with humans can take many forms, varying not only in relation to the physical mode of communication but also the purposes of the interaction. In some cases artificial agents will require a deep understanding of human cognition and motivation, e.g. in some forms of intelligent teaching systems (Simon 1967; Sloman 1992; Picard 1997).

It is to be expected that the toolkit requirements for such varied applications will be very diverse. Any general purpose toolkit should at least provide means for simulating the application environment if it is different from the development environment.

If there is an external environment the toolkit should provide "hooks" for connection to the final sensors, motors, interaction devices, etc. If sensors such as TV cameras and microphones are to provide information about the environment there will generally be considerable complexity in the perceptual processing, including many levels of interpretation (edge features, optical flow patterns, texture regions, 3-D structure fragments, recognised objects, perceived spatio-temporal relations between objects, perceived functional relationships, recognised larger contexts, e.g. a seminar, a party, a family dinner, etc.). This may require much use of prior domain knowledge in "top down" processing to deal with ambiguities in the data.

This paper does not fully address perception in a physical environment, and from now on will mainly be concerned with development of simulations, though with increasing sophistication of virtual reality environments the differences between a simulation and a physically realised system may diminish.

Often simulations will use software which is not part of the agent development environment, but a separately developed specialised simulation tool (e.g. flight simulators, simulators for various kinds of machinery, robots, etc.). However if there are frequently used simulation environments, some useful abstractions may be possible, leading to useful libraries for an AI toolkit (e.g. packages for simulating 2-D motion on a horizontal surface, packages for handling natural language interaction with a user, packages for simulating widely used robots).

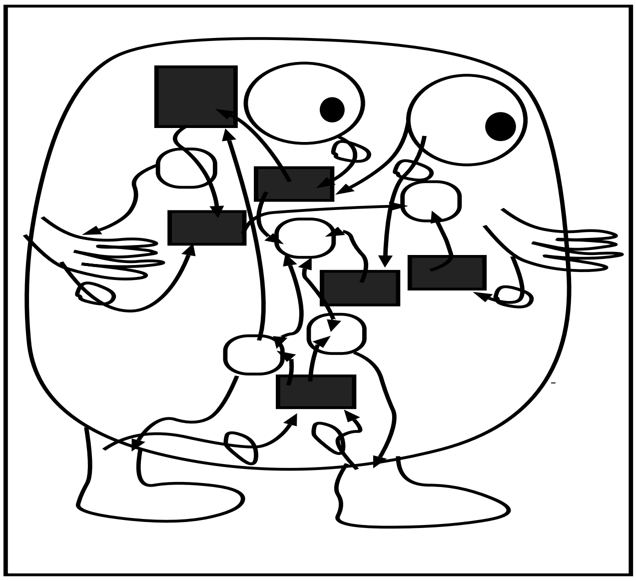

Figure 2: **Multi-process agents**
(Here rectangles represent short or long term databases and ovals represent processing units. Arrows represent flow of information.)

Often a simulated environment will be a useful step towards physically (and socially) embedded final scenarios, provided that adequate care is taken not to oversimplify the simulation and not to overgeneralise results of simulations.

Inside One Agent

For research which aims (for scientific or for engineering purposes) to model human or animal intelligence there will be a need to design agents whose internal architecture is extremely complex: in some cases at least as complex as the kinds of multi-agent scenarios depicted in Figure 1. As indicated (crudely) in Figure 2 there may be various sensors and motors connected to a variety of internal processing modules and internal short term and long term databases, all performing various sub-tasks concurrently, possibly subject to unpredictable interruptions or distractions from one another and from the environment.

So a general toolkit must support not only multiple agents which act concurrently and asynchronously, but also within-agent components which act concurrently and asynchronously *within* individual agents, performing multiple different tasks (Minsky 1987). I.e. if a typical discrete event simulation system is provided, it must support something like a hierarchical structure in which concurrent modules themselves contain concurrent modules.

If different goal generators and other mechanisms act concurrently in an agent, various kinds of inconsistency can arise, at different levels: conflicting motor commands, conflicting goals, conflicting processing resource requirements, etc. To deal with this the toolkit library may have to include a variety of different conflict resolution mechanisms.

AI research over several decades shows that different *types* of mechanisms will be required for different components, including rule-based reactive systems, neural nets, parsers, meaning generators, sentence generators, pattern-directed associative knowledge stores, low level image analysers mainly crunching numbers, high level perceptual mechanisms mainly manipulating structures, simulations of other agents, etc. This in turn probably imposes a requirement for using different kinds of language for specifying and implementing different sub-mechanisms.[1]

So, we need toolkits which can support scenarios in which there are not only multiple agents and objects all processing information and performing actions concurrently, but also multiple sub-mechanisms within such agents, acting with varying degrees of coherence. We now try to give more details, starting with simpler systems and gradually adding complexity.

The need for concurrency WITHIN agents

Concurrent more or less asynchronous sub-components of a human-like agent may include processes as diverse as:
- taking in and interpreting new perceptual information,
- processing new communications from other agents,
- generating new motives,
- comparing motives in order to decide which to adopt as intentions,
- deciding whether to abandon or revise existing intentions,
- formulating and comparing possible plans to achieve intentions,
- deciding which plan (if any) to adopt
- executing plans,
- monitoring action performance,
- deciding whether to revise plans, or postpone execution,
- generating linguistic communications,
- monitoring and evaluating internal processes

and many kinds of learning and adaptation.

The processes listed here may be described as "deliberative" processes. It is also possible to have an architecture composed entirely of "reactive" mechanisms implemented on fast, dedicated parallel circuitry, with no planning capability (though innate plans may be used.) That requires the designer (or evolution) to anticipate and pre-design all possible types of plans required for all possible contexts, and may even require an impossibly large memory to store all those plans. Including a deliberative mechanism allows plans to be constructed as and when necessary, e.g. using symbolic AI planning methods, reducing the need for prior design (or evolution) of behaviours and also possibly reducing storage requirements. These trade-offs are not always understood when supporters of reactive behaviour-based systems claim that deliberative capabilities (good old fashioned AI techniques) are not needed.

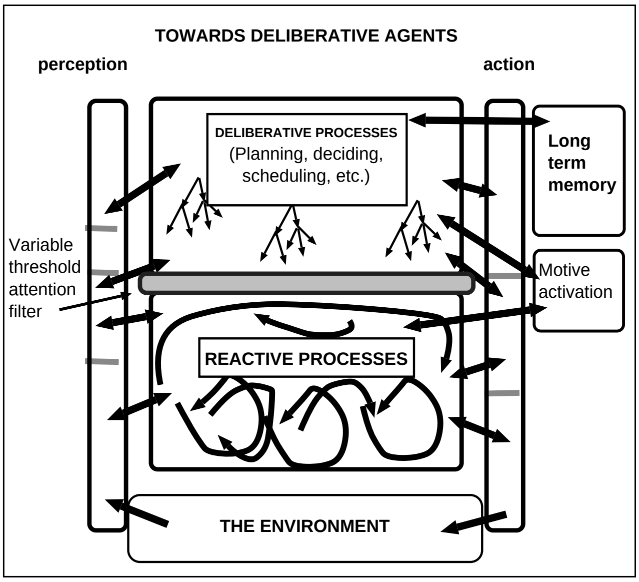

Figure 3: **A hybrid system combining reactive and deliberative mechanisms.**
(Various supporting mechanisms are needed, e.g. a long term associative store for use in planning and prediction, a representation for goals, an "attention filter" to protect urgent and important resource-limited deliberative processes etc.)

Figure 3 sketchily indicates a hybrid architecture which includes interacting deliberative and reactive mechanisms, all running in parallel with perceptual and motor control systems, goal generating mechanisms, and a long term associative memory (required for deliberation, in order to answer questions about what actions are possible in particular contexts, what the consequences of those actions would be, and how objects in the environment might react).

Alarms and variable resource allocation

Some internal processes may be relatively slow (e.g. deliberation, perception of complex unfamiliar structures picking up a delicate and fragile object. If items in the environment can be fast moving (predators,

[1] There is a methodological stance which claims that naturally evolved animal brains have no intelligible modular structure, and therefore comparable systems cannot be designed by human engineers. A paper challenging this argument is in preparation. I believe evolution "discovered" the same sorts of requirements for modularity of design as human engineers.

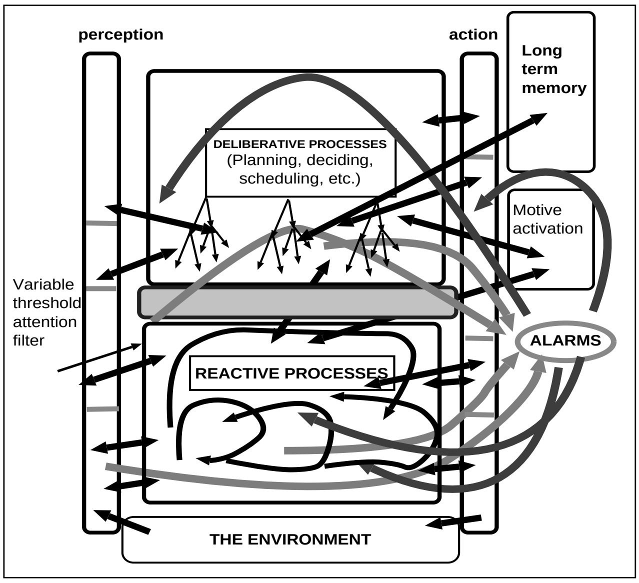

Figure 4: **The previous figure with global alarm mechanism added.**
(The alarm system, a special purpose reactive mechanism, receives inputs from all other parts of the system, uses rapid pattern recognition to detect opportunities, dangers, etc. and reacts by sending "powerful" control signals to all parts of the system.)

prey, or random projectiles) it might be useful to have an additional trainable "global alarm" system operating in parallel with the routine reactive and deliberative mechanisms, able to detect the need for rapid action or redirection of attention or freezing, etc. Such an architecture is depicted loosely in Figure 4. It seems that humans and other animals include one or more such mechanisms, which, among other things, can produce emotional reactions.

This illustrates an important point, namely that it may be necessary to distinguish components which operate at different speeds. A toolkit should make it possible to explore and compare an agent which has a very fast planning mechanism with one which has a much slower planning mechanisms, and then investigate the effects of adding "anytime" planning facilities and alarm systems. This requires user-definable resource-allocation mechanisms for components of the architecture (one of the features of the SIM_AGENT toolkit we have developed).

Another requirement, if alarm systems are to be able to work, is that one mechanism should be able to interrupt or redirect another, requiring provision for something like signal handlers or event handlers, throughout the system.

Internal monitoring: meta-management

For reasons which have been discussed at length elsewhere (Beaudoin 1994; McCarthy 1995; Sloman 1997; Wright, Sloman, & Beaudoin 1996) it may be desirable in some kinds of agents (especially agents with advanced forms of human self-consciousness), to introspect, i.e. monitor internal states, categorise them, evaluate them and attempt to control them.

E.g. an agent which learns that in context A, reasoning strategy S1 often leads rapidly to a good conclusion, whereas in context B strategy S2 works better, can use recognition of the context to select a reasoning strategy. Without meta-management this would have to be directly programmed into the deliberative mechanism from the start, rather than being something the agent can discover for itself.

If the meta-management architecture also supports inspection of intermediate structures in perceptual mechanisms this will enable agents to compare differences in how the same things look to them. In human agents this is also relevant to learning to draw things realistically. Meta-management may also be a mechanism whereby agents share a culture: for instance if ways of thinking and deciding are favoured in a culture then individuals may learn to use them. An example might be learning to prefer unselfish decision making criteria to selfish ones.

A crude depiction of an architecture including this sort of functionality in addition to the previous kinds is shown in Figure 5, from which the alarm mechanism has been omitted to save clutter.

It is not clear how many animals can direct attention inwardly in this sense, and categorise, evaluate and control internal processes, though humans clearly can to some extent. I do not know whether a rat or a chimpanzee can control its thought processes.

In humans, control of attention is partial: in certain emotional states such as grief or excited anticipation, for instance, we may find our thoughts wandering away from the task in hand to a topic related to the source of the emotion. Where part of the task of an agent designer is to simulate these emotional processes a multi-layer hybrid architecture may be needed.[2]

In artificial agents there could also be various kinds of interrupt mechanisms which interfere with such control, for instance a type of global alarm system which can use very rapid pattern matching to decide to redirect processing throughout the system (as the limbic system seems to be able to do in humans and other animals). There might be different global alarm systems of different complexity, operating at different speeds.

The need for speed would rule out deep analysis and reasoning. Consequently there is a serious risk that such a system will make mistakes and interfere with complex tasks. However turning off the alarm system could be dangerous as urgent and important opportunities and dangers might then be missed.

[2]For a recent widely-cited analysis of the role of emotions in intelligence see Damasio(1994).

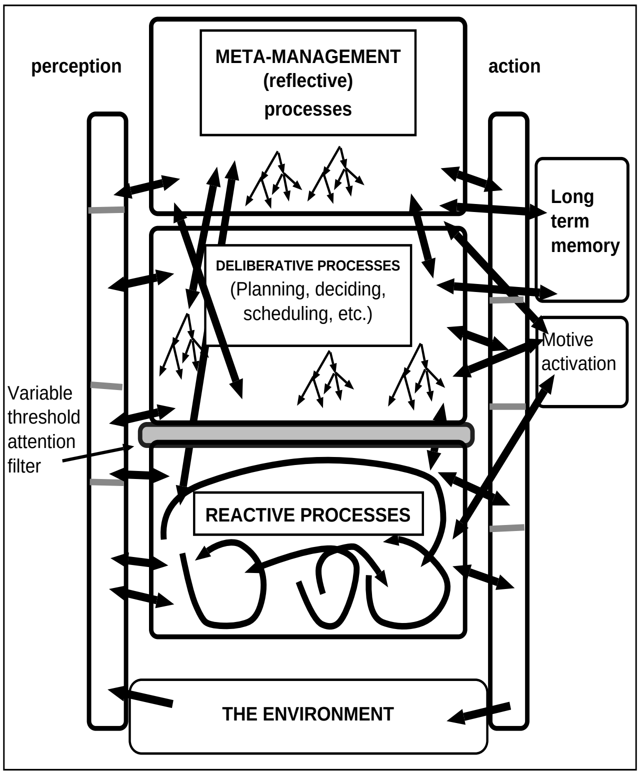

Figure 5: **Towards a human-like reflective architecture: the previous system with mechanisms able to monitor, categorise, evaluate and modify processes in other systems.**
(A "global alarm" mechanism (like an octopus with tentacles reaching to and from all other parts) could be added, as in previous figure. It is possible that metamanagement mechanisms evolved by copying and then redesigning earlier global alarm mechanisms.)

Re-building the Boat

Consider sailors re-building a boat while sailing in it, until it no longer has any of its original parts. I suspect something similar happens between the birth of a human infant and normal adult development.

There may also be a practical need for artificial agents to add or remove components in their architecture, e.g. buying new "plug-ins".

Tools to allow various types of self-reconstruction will be required for building such systems. I.e. some development tools may be required at run time.

Varieties of learning and development

The more complex an agent's architecture is the more scope there is for adaptation, development or learning. For instance in Figure 5 kinds of change could include addition or removal of sub-components in the architecture, addition or modification of links between components, development of new formalisms for use in various sub-mechanisms, addition of new facts in the long term memory used by the deliberative layer, modification of filtering strategies, development of new ways of categorising internal states, development of new deliberation strategies, and fine tuning of sensory and motor systems.

If a global alarm system is added, then it may be able to learn new patterns requiring fast global reactions, and it may also be able to modify the reactions associated with various patterns over time. E.g. an adult learns to suppress reactions that are useful in childhood because they inform a parent of one's needs.

Development Environments

It has proved quite difficult to design and implement agents with the degree of complexity and flexibility outlined here. One reason is the difficulty of knowing what sort of design is required. This suggests a need for tools to explore possible designs and design requirements (e.g. by examining how instances of first draft designs succeed or fail in various domains and tasks). I.e. support for very rapid prototyping is helpful. This raises the issue of the familiar trade-off been compile time checking and optimisation *vs* flexibility.

Many agent toolkits are geared to support a particular type of agent (e.g. agents built from a collection of neural nets, or agents which all have a particular sort of cognitive architecture such as the SOAR architecture proposed by Newell, or the ACT-R architecture proposed by Anderson). While these may be useful for exploring the implications of particular theories, as *general purpose* toolkits they are inadequate. The field of AI in general, and the study of systems of interacting agents in particular, is in its infancy: we have very little idea of what the full range of information processing engines and architectures might be. For instance, brain scientists are continually discovering new roles for neurotransmitters in animal brains, and there may be profound surprises ahead. Thus a toolkit committed to any particular architecture is likely to impose a premature commitment, discouraging the investigation of alternatives. (This is not to deny the usefulness of such toolkits for their domain of application.)

We also need a type of toolkit which not only supports the kind of complexity described above, but which does not prescribe any particular architecture for agents, so that we can learn by exploring new forms. It is not clear whether any single toolkit can suffice for this purpose, or whether a collection of different toolkits will always be needed. It is very unlikely that one will suffice for everything, but we can try to find a minimal set which will cover the known classes of requirements.

An analogy: although almost all programming languages are inherently general (since they can all be used to implement a general purpose Turing machine) there is no one which has proved to be best for all purposes. However, some (e.g. Common Lisp, Pop-11) provide support for a wider range of programming styles than most others (e.g. procedural, functional, list-processing, pattern based, object oriented styles), and through the use of such things as interpreters or incremental compilers, automatic garbage collectors, powerful syntax-extension mechanisms, and integrated editors they are also better suited to certain types of rapid exploratory prototyping.

However one requirement for any toolkit is that it should support the re-use of components of previous designs so that not all designers have to start from scratch. Making this possible involves producing a well-annotated "library" of components, algorithms, forms of representation, along with tools for assembling them into architectures.

Ideally a toolkit should support both explicit design by human researchers and also automated design of agents e.g. using evolutionary mechanisms, or mechanisms which can take a high level functional specification for an agent and work out which components are needed. It is very likely that this cannot be fully automated, so interactive co-operative design tools will be required.

Combining paradigms

Since different kinds of mechanisms are required for different subsystems, one of the important tasks in designing a general purpose toolkit is to find ways of combining the different mechanisms. Unfortunately researchers instead often argue about which mechanisms are the "right" ones!

A condition-action rulesystem may include some conditions which have to be matched against a very large long term memory with generalisation and interpolation capabilities, which might be implemented as a neural net. We need to continue exploring techniques for combining such capabilities. This is done in Anderson's ACT-R system (Anderson. 1993). A different mechanism for integrating condition-action rules with "subsymbolic" mechanisms can be found in our SIM_AGENT toolkit (Sloman & Poli 1996). No doubt there is a wide variety of types of synthesis to be explored.

Designing physical robots often forces use of different paradigms for different components, e.g. low level and high level vision, fine-grained motor control, various kinds of symbolic and neural learning mechanisms, and even integration between mechanical and software control, for instance in the use of a "compliant" wrist to guide cylinders into tight-fitting holes, etc. There is no reason why similar eclecticism should not be useful in systems which inhabit software worlds, and designers of general purpose toolkits should be aware of the need to support this.

Windows into the souls of agents

When agents are very complex it can be very difficult to understand what is happening when they run, especially if the observed behaviour is unexpected and debugging is required. One obvious solution to this problem is to provide support for trace printing. However experience with such systems shows that it is very easy to generate megabytes of unusable trace printing if there are several agents each with multiple complex internal processes.

A partial solution is to use graphical displays showing the interactions of the agents. This is particularly useful, indeed essential, where the interactions include simulated spatial motion. However it should also be possible at run time to select individual agents, and explore their internal processes either by selectively turning various kinds of textual and graphical tracing on and off, or by making the system pause so that one can probe the agent's architecture, analyse the contents of its data-bases, neural nets, etc. This will often require graphical tools.

Such tools are easier to provide when all agents have similar architectures and environments: where a general purpose toolkit supports a wide range of architectures, users will have to be able to extend the graphical interrogation mechanisms to suit the architectures and environments: often a non-trivial task.

In some cases the internal structure is too complex to inspect directly. In that case, it may be necessary for the agent to engage in a dialogue with the user describing its current beliefs, desires, plans, intentions, fears, etc. This will require self-monitoring capabilities as well as something like natural language processing mechanisms. Speech synthesisers may also be useful.

What Are the Scenarios to be Used For?

There are large differences between scenarios which are used entirely for the purposes of theoretical research, e.g. to test out models of human cognition, or ideas about agent architectures or types of communication formalisms, and scenarios which are used for a practical application.

In particular for theoretical work many insights often come from incomplete implementations, since the process of implementation can reveal gaps and errors in the ideas being developed.

Practical applications also vary. For instance there is a difference between the design of a disaster control system which is to be used to control real people and equipment in dealing with earthquakes or fires or ship collisions, and a disaster control system which is used only in testing disaster management strategies, or in training environments for the people who will themselves later have to take responsibility in real situations.

In some ways the training context is more demanding since the system not only has to be able to perform, it also needs to be able to explain and justify its performance and also to analyse and comment on the performance of a trainee. This requires a kind of self-knowledge which may not be relevant for all fielded applications. Of course in some applications, providing explanations to human operators may be included in the task requirements.

Additional selection criteria

Besides the issues already discussed, there are additional factors which may influence the choice of a toolkit. Some of these are listed here.

1. To what extent are the objectives and design strategies already well understood in advance of use of the toolkit?

Where a design specification for production of one or more intelligent systems is well defined in advance, it may suffice to produce tools which include a precisely defined formalism for expressing designs along with compilers or interpreters which can animate such a design specification.

In that context it also makes sense to require tools to verify formally that the animation meets some formally specified set of requirements.

Where the problem is not well defined in advance and part of the objective of the construction of a new system is to explore what the design requirements are and to work out which kinds of designs might help to meet different sorts of requirements, then tools which make exploration and experiment relatively easy may be more important than tools which accept mathematically precise formal specifications and which automate verification.

Use of logical specification will be most useful where requirements and designs are well understood in advance and formal analysis of systems is required, whereas object oriented programming with multiple inheritance and generic functions (multi-methods) will be more useful for exploratory research. In the latter case a toolkit supporting a variety of programming paradigms may be helpful, especially if the exploration involves designing the simulation environment as well as many different components in the agents, including perceptual and motor mechanisms.

2. Which is more important: ease and speed of development and debugging, or speed of execution of the developed system?

AI languages usually have many features which make them very suitable for interactive development and testing. These include dynamic scoping of identifiers, interpreted or incrementally compiled code, availability of symbol table and compiler at run time, use of indirection via identifiers in a global symbol table, so that redefining the value of a variable causes old code to point (indirectly) to the new value, etc. For example typically in an AI environment (Lisp, Prolog, Pop-11, Scheme), while a system is up and running you can in a fraction of a second recompile a redefined rule or procedure and everything which used the old definition will thereafter automatically use the new one, via its name: there is no need to re-link and start again. The space previously used by the old one will be automatically reclaimed by a garbage collector, preventing memory leaks. All this can lead to extremely rapid exploratory development and testing.

Moreover, these languages support *schematic* re-usable high order functions with minimal constraints, such as a sorting procedure which takes an untyped list and an untyped ordering predicate and uses a general sorting algorithm (such as merge sort) without caring whether the list elements are all of the same type as long as the predicate produces a boolean result when applied to any two of them. This kind of flexibility makes it very easy to re-use powerful general purpose libraries including databases, pattern matchers, etc. in contexts which would be very awkward for more "disciplined" languages.

Such facilities are often ruled out in strongly typed languages (even incrementally compiled ones like ML) because the typing is used to perform optimisations which are not guaranteed to be valid if only one part of a complex previously compiled collection of procedures is changed.

Another feature of many AI environments is the use of an interpreter which can be told at run time to modify its behaviour so as to give the users more tracing information or the ability to have closer control over running.

Dynamically recompileable methods in an object-oriented development environment also provide support for this type of flexibility.

People who are used to languages like C and C++ and have never used AI languages often do not appreciate the advantages of such flexibility.

An extreme example of an ill-defined problem might be the task of making some sort of intelligent interface easy for a certain class of users. If we don't initially have a precise specification of users' cognitive mechanisms, we can have no clear idea what kind of appearance or behaviour will or will not be easy. If we have some "hunches" regarding ease of use, we can implement them and then try the system out with real users. A flexible development environment should make it easy rapidly to try making various modifications, including modifications suggested by a user, to see if the user likes them or not, without having to recompile everything. Not all these modifications will be in a parameter space that has previously been found relevant. It may be necessary to redefine some procedure or add new procedures to make the changes proposed, e.g. changing a help procedure so that it provides a new panel of options.

Provision of facilities for easily changing code or the behaviour of modules at run time without having to restart the whole system often introduces bugs and inefficiencies. Nevertheless they are exactly the sorts of facilities which can make development go very much faster when the problem is ill defined and when a great of experimentation is still required to understand the problem. Moreover the flexibility which sometimes leads to bugs also supports very flexible and powerful debugging (e.g. adding a one-off procedure at run time to search a data-structure suspected of being corrupted).

Even if facilities supporting rapid prototyping and exploration of options are incompatible with optimising run time and space performance, it is arguable that most interesting research areas in AI are sufficiently ill-defined that toolkits which optimise flexibility rather than performance will be needed.

Fortunately changes in hardware have been providing exponentially increasing performance and available memory over the last two decades. This means that hardware developments have led to far greater speed increases than software modifications can (including development of new languages or new compilers). Perhaps this will continue.

3. Is it essential that the developed system be provably correct as in some safety critical applications?

One consequences of the kind of flexibility outlined in the previous section is the impossibility of verifying that a system will meet some specification. E.g. the use of un-typed identifiers add considerable flexibility to many AI systems, but at the same time can make it difficult or impossible to *prove* properties of programs.

Furthermore if users have the freedom to redefine things at run time in the manner required for exploratory design and rapid prototyping, then in some cases exactly the same freedom may have to be given to systems which adapt themselves to new users, new contexts or new tasks. Such self-modifying code may be required for agents which begin to match human intelligence and flexibility. In more mundane contexts self-modifiability may be a requirement for systems which can expand themselves by acquiring new "plug-in" modules at run time (by analogy with tools like Netscape).

For all these reasons it is arguable that the sorts of flexible development tools which are required for much interesting AI research would be incompatible with the *very* restrictive requirements typical of safety critical systems.

On the other hand we often include humans as components in safety critical systems (e.g. human pilots) and it may be that for similar reasons we may eventually have to tolerate intelligent artificial agents which are no more provably correct than humans are.

Challenges for Theorists

This analysis of the design and development problems for intelligent systems raises a number of problems which may be disconcerting for theorists.

• The design of systems of such complexity poses a formidable challenge. Can it be automated to any useful extent? (In simple cases genetic algorithms and similar techniques may be relevant.)

• Will we ever be able to automate the checking of important features of such designs?

• It seems likely that the sort of complexity outlined above will be required even in some safety critical systems. Can we possibly hope to understand such complex systems well enough to trust them?

• Do we yet have good languages for expressing the *requirements* for such systems (e.g. what does "coherent integration" mean? What does "adaptive learning" mean in connection with a multi-functional system?)

• Do we have languages adequate for describing *designs* for such systems at a high enough level of abstraction for us to be able understand them (as opposed to millions of lines of low level detail)?

• Will we ever understand the workings of systems of such complexity?

• How should we teach our students to think about such things?

Conclusion

I have tried to indicate some of the diversity and complexity to be found in agent design tasks, and have spelled out some ways in which this impinges on requirements for agent toolkits.

Some of the complexity comes from the rich ontologies in the environments of agents, especially where the environment contains other kinds of agents of varying types. Some of it comes from the fact that agents may themselves have a rich and complex internal architecture consisting of many concurrent interacting modules of various types, including reactive modules, deliberative modules, sensory analysis and interpretation modules, motor control modules, and reflective modules capable of monitoring, evaluating, and to some extent controlling other internal processes.

Apart from the inherent complexity of the designs to be created, further complexity can arise when the design task is not initially well specified and is in part a research process, exploring different architectures and environments in order to gain an understanding of the space of possible designs. In that case it may be possible to facilitate the exploration by using a language which supports rapid prototyping and incremental development and testing. In other cases tools for automatic adaptation or evolution may be helpful.

As the systems being developed grow more complex, various support tools will be needed, including a library mechanism for saving re-usable components, subarchitectures, algorithms, and forms of representation, along with indexes and other aids for selection and assembly of these items. Re-usable graphical and other tracing and profiling mechanisms which can be selectively turned on and off asynchronously will also be useful.

In some cases we may end up with systems which are too complex for us to understand fully, especially if they have developed using self-modification mechanisms. In such cases we may simply have to trust the artificial agents, much as we trust other people, at least after they have been tested and proved trustworthy.

One of the main points to be stressed in this analysis is that we do not yet know what sorts of architectures will be needed for intelligent systems of the future, whether they are practical systems or research models. Consequently it is hard to predict what sorts of tools will be required. It may even turn out that for some tasks new kinds of information processing engines will be necessary including for instance, chemical mechanisms or quantum computers. In the meantime we should not unduly restrict our research horizons by using only toolkits designed to support a fixed class of architectures.[3]

Acknowledgements

This paper includes ideas from: Riccardo Poli, Brian Logan, Luc Beaudoin, Darryl Davis, Ian Wright, Jeremy Baxter (DERA), Richard Hepplewhite (DERA).

The work is partly funded by grant no CSM3202710 from DERA Malvern.

References

Anderson., J. R. 1993. *Rules of the Mind.* Mahwah, NJ: Lawrence Erlbaum Associates,.

Beaudoin, L. 1994. *Goal processing in autonomous agents.* Ph.D. Dissertation, School of Computer Science, The University of Birmingham.

Damasio, A. R. 1994. *Descartes' Error, Emotion Reason and the Human Brain.* Grosset/Putnam Books.

Davis, D. N. 1996. Reactive and motivational agents: Towards a collective minder. In Muller, J.; Wooldridge, M.; and Jennings, N., eds., *Intelligent Agents III — Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages.* Springer-Verlag.

Logan, B. 1998. Classifying agent systems. In Logan, B., and Baxter, J., eds., *Proceedings AAAI-98 Workshop on Software Tools for Developing Agents.* Menlo Park, California: American Association for Artificial Intelligence.

McCarthy, J. 1995. Making robots conscious of their mental states. In *AAAI Spring Symposium on Representing Mental States and Mechanisms.* Accessible via http://www-formal.stanford.edu/jmc/consciousness.html.

Minsky, M. L. 1987. *The Society of Mind.* London: William Heinemann Ltd.

Newell, A. 1990. *Unified Theories of Cognition.* Harvard University Press.

Picard, R. 1997. *Affective Computing.* Cambridge, Mass, London, England: MIT Press.

Simon, H. A. 1967. Motivational and emotional controls of cognition. Reprinted in *Models of Thought,* Yale University Press, 29–38, 1979.

Sloman, A., and Logan, B. S. 1998. Architectures and tools for human-like agents. In Ritter, F., and Young, R. M., eds., *Proceedings of the 2nd European Conference on Cognitive Modelling,* 58–65. Nottkingham, UK: Nottingham University Press.

Sloman, A., and Poli, R. 1996. Sim_agent: A toolkit for exploring agent designs. In Wooldridge, M.; Mueller, J.; and Tambe, M., eds., *Intelligent Agents Vol II (ATAL-95).* Springer-Verlag. 392–407.

Sloman, A. 1992. Prolegomena to a theory of communication and affect. In Ortony, A.; Slack, J.; and Stock, O., eds., *Communication from an Artificial Intelligence Perspective: Theoretical and Applied Issues.* Heidelberg, Germany: Springer. 229–260.

Sloman, A. 1994. Explorations in design space. In *Proceedings 11th European Conference on AI.*

Sloman, A. 1996. Beyond turing equivalence. In Millican, P., and Clark, A., eds., *Machines and Thought: The Legacy of Alan Turing (vol I).* Oxford: The Clarendon Press. 179–219. (Presented at Turing90 Colloquium, Sussex University, April 1990. Also Cognitive Science technical report: CSRP-95-7).

Sloman, A. 1997. What sort of control system is able to have a personality. In Trappl, R., and Petta, P., eds., *Creating Personalities for Synthetic Actors: Towards Autonomous Personality Agents.* Berlin: Springer (Lecture notes in AI). 166–208. (Originally presented at Workshop on Designing personalities for synthetic actors, Vienna, June 1995).

Wright, I.; Sloman, A.; and Beaudoin, L. 1996. Towards a design-based analysis of emotional episodes. *Philosophy Psychiatry and Psychology* 3(2):101–126.

[3]The SIM_AGENT toolkit was designed to support exploration of a very wide range of architectures. It is described briefly in Sloman & Poli (1996) and at the following web page, which also points to the source code: **www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html**