Autonomous Agents: Characterization and Requirements

Jose C. Brustoloni

November 1991
CMU-CS-91-204

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

A fresh assessment of what autonomous agents are or may aspire to be is presented in the paper. We dismiss the traditional dichotomy between reactive and symbolic architectures. A classification in terms of the amount of knowledge embedded in the system is made, instead, running from *regulatory* to *planning* to *adaptive* agents. Regulatory agents are implementable as *control systems* or *automata*, and represent the ideal case in which the agent has all the knowledge it needs. We find that each type of agent is best suited for a particular kind of behavior. The different kinds of behavior form a *hierarchy* in terms of frequency of occurrence and response time, and do not gracefully reduce to each other. This suggests implementing agents as hierarchies of simpler agents, each specialized in the kind of behavior it implements. In this scheme, agents at one level act through agents at the next lower level. We introduce the concept of *drive* to address the issue of where an agent's goals come from. A discussion is made of common pitfalls and suggestions for the design of planning agents. Finally, a *parallel* architecture that embodies many of the ideas presented is sketched and discussed. This architecture solves the two major problems present in Georgeff's earlier architecture, PRS: strictly sequential execution, and inability to learn.

This research was sponsored by the Office of Naval Research under Contract N00014-86-K-0678.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

Keywords: artificial intelligence, integrated architectures, intelligent agents, real-time AI, problem solving, planning, procedural reasoning, case-based reasoning, machine learning, drives, behavior synthesis, parallel execution

1. What is an autonomous agent (and what is it not)?

Autonomous agents are systems capable of autonomous, purposeful action in the real world. We deliberately avoid the more common expression "intelligent agent", which inevitably leads to the itchy question: what is intelligence, anyway? The amount of intelligence to which a particular behavior is ascribed depends to a large extent on the observer of that behavior. And Brooks' robots [3] demonstrate that it is possible to build interesting and useful agents that, nevertheless, lack much of what one would consider "intelligence".

Intelligent or not, autonomous agents must be able to perform in the real world. Agents must be *reactive* – that is, be able to respond to external, asynchronous stimuli in a timely fashion. Whatever the computation or signal processing involved, it must be real time. The data input and output subsystem must be carefully matched to the task to be performed – adequate sensors and effectors must be provided, and go a long way in bringing about the agent's functionality. Attempts to skimp on the I/O and compensate for it with more computation are normally ill-fated. The data rates are such that a less than clever I/O design rarely meets the real time constraints. Attempts to generalize on the sensors and effectors, trying to obtain a more flexible agent, can also add formidable difficulties – it is hard enough to devise a more constrained set. And, if agents are tools, there is no reason why one can't have lots of them, one for each purpose – much as we have hammers and screwdrivers and do not expect one to do the other's job.

The fact that agents are to perform purposeful action is often interpreted as meaning that they are goal-directed. Actually, this is an oversimplification. A better interpretation is that agents have *drives* (in a somewhat "psychological" sense), and devote their resources to satisfying these drives [2, 9]. In doing so, one can observe purpose in their actions. Many different goals may be attempted, any of which would satisfy a given drive; or maybe the drive is already temporarily satisfied, in which case the agent had better occupy itself with something else. For example, the drive to avoid hunger may lead to the goals of going to a restaurant and eating, or of cooking and eating, or to no goal at all, if the agent is already full. Goals have a merely instrumental function; what should ultimately direct the agent are its drives.

Further along these lines, there actually is a *hierarchy of actions and goals* in an agent, with drive satisfaction at the top. Actions and goals at one level exist only to accomplish goals at a higher level, and can generally be replaced by other actions and goals, which also would attain the higher level goal. At the bottom of this hierarchy are the primitive actions – the elementary actions directly supported by the architecture and from which more complex actions are composed.

Drives are not necessarily linear and consistent; they may compete for resources and spring incompatible goals. A major concern is how to coordinate the utilization of resources in a reasonably "fair" way, avoiding "starvation" of this or that drive.

And, while an instrumental goal is being pursued, it may be necessary to temporarily "turn off" a drive, lest precipitation might spoil the overall plan (an instance of the "reality principle" of Freudian psychology). This raises the need to adopt some kind of *regulation matrix*, controlling the opportune manifestation of drives.

If agents are to be autonomous, they have to either *know* or else be able to *find out* how to satisfy drives and achieve goals. "Find out" generally means planning – search for a functionally equivalent schedule of simpler actions. The duality of knowledge and search is a long established fact in AI [21] – you don't have to search if you know the answer already, but if you don't, you can search for it. However powerful, search requires a lot of non-trivial, hard to acquire knowledge. In the context of agents, at a minimum a knowledge of the primitive actions is required, along with what is required for them to be applicable, and their consequences. Based on these, the agent can search for composite actions that will achieve desired consequences, which no single primitive action would be able to.

Knowledge may be embedded *structurally* or *symbolically*. In the first case, the architecture itself fixes the processing of stimuli and generation of actions. Normally, no search is involved. Examples of this approach are neural networks [13] and subsumption architectures [3]. In the second case, stimuli are transformed into symbols; the agent manipulates these symbols and determines appropriate actions through these manipulations. PRS [10, 11] and Robo-Soar [17] both use symbolic representations. Being more direct, structural representations often obtain better reactiveness/performance. Symbolic representations, on the other hand, allow for more flexibility – the same architecture may process stimuli very differently, if programmed to do so. Symbolic representations also tend to make it easier to experiment with and implement more complex functionality.

Although often debated, there is little of fundamental in this dichotomy between reactive/structural architectures and symbolic architectures. Stimuli are always transformed into some kind of internal representation – be it an electric pulse, the voltage in a point of a circuit, or a string of bits or characters. The situation is similar to the distinction between hardware and software: any kind of software can be implemented in hardware, for performance; and any kind of hardware can be turned into software (of a fast enough machine), for experimentation and flexibility. They are functionally equivalent.

In all but the simplest agents, one can expect to find a considerable amount of knowledge embedded structurally in the I/O subsystem. The sensors have to "know" (that is, architecturally ease and favor) the agent's task.

2. Types of agent

2.1. Regulation agents

Regulation agents are those that don't do planning. They have built-in knowledge to cope with whatever stimuli they may face. They have been architected in such a way that they always know what to do, and such prescribed actions satisfy the overall objectives of the agent [22, 4, 15]. An example of such an agent is the thermostat controlling the temperature in a house. Evidently, that's an autonomous system operating in the real world, and it does act (turn the furnace on and off) with a satisfactory sense of purpose (maintain the temperature). Another example: the interlock system in a railroad (the complex system of switches and relays built to avoid train crashes). Yet another example: Brooks' robots [3]. And a final example: distributed control systems (DCS) used in oil refineries (networks of controllers and operator stations used to control the various flows, pressures, temperatures

and other variables in the plant).

As these examples demonstrate, regulation agents admit structural (thermostat) or symbolic (DCS) implementations. In either case, performance is orders of magnitude better than if planning were involved. These systems can be made *tunable* to the dynamics of the process they control and, sometimes, can also be made *configurable* – that is, the type of behavior to be exhibited may be selected from several pre-defined options. A careful match of I/O is always mandatory, though.

In principle, any planning can be avoided by knowledge (of a corresponding complete plan). Therefore, any agent should be able to be turned into a regulatory agent, with performance advantages. In practice, however, this cannot always be done. The problem, often, is the immense variety of situations and goals that may be encountered, making it impractical to pre-specify for each the corresponding correct plan. Also, errors often occur in the execution of plans; a complete pre-analysis of the appropriate recovery in each situation is frequently an even more distant possibility.

On the other hand, if actions reliably obtain their expected consequences, the set of possible goals and plans is relatively small, and it is possible to pre-analyze the relative importance of different goals and desirability of alternative plans, then definitely a regulation agent (control system) is the implementation of choice.

2.2. Planning agents

Planning agents, aside from what a regulation agent does, are also able to plan. As seen above, there are situations in which planning is well-justified.

The problem with planning is that it is terribly expensive [7]. It is very hard to do in real time. Contrary to control systems and automata, there is no well developed, generally accepted way to go about planning. Several alternative approaches exist:

- The problem solving paradigm of AI;
- The case-based paradigm of AI;
- The various methods of operations research (OR);
- Various randomizing algorithms.

These algorithms are in fact so different that agents based in them should actually be called "problem solving agent", "case-based agent", "OR agent", or yet "randomizing agent", respectively.

The problem solving approach begins [20, 6, 18, 16] by modeling each primitive action in terms of its required pre-conditions and expected results. Using these models, a search is done for a sequence of actions such that the first action is applicable in the current state of the world, and the last action's results include the goal. The plan may be required to be optimal or almost so, or more commonly not (satisficing is enough).

A number of mechanisms have been proposed to remember the plan, or some generalization of it, so that the next time the same problem is encountered no planning needs to be done. Among these: explanation based learning (EBL) [19, 6], chunking [16], caching [18].

The main problems with problem solving are:

1. It is expensive. (But then, you don't have to plan more than once for the same problem.)
2. An accurate and, worse, complete model of the available actions and of the world must be maintained – and this is very hard to get right.
3. It has been proposed as a completely general approach [20, 16] for artificial intelligence, and, while the argument is defensible, there certainly are myriad situations for which it is not particularly well suited (e.g., problems that do not require planning; problems involving common sense; situations in which a solution can easily be demonstrated, but the elements that lead to it are difficult to formalize – can you imagine the model of a painter or basketball player?)

The case-based approach [12] consists of keeping a library of cases – plans already tried and known to work. When a problem is found, the library is looked up to find an applicable pre-tested plan. This plan may not match the problem exactly, in which case some tweaking may be necessary. The idea is solve the problem by analogy, rather than search. It is generally expected that analogy would be a lot cheaper than search. The problem is that a satisfactory computational model of analogy has not yet been found. For example, how do you retrieve the solution to the "most similar" problem in the library? What is "similar", anyway? There is a danger here of getting into something very much like an expensive search. This approach has not been studied nearly as much as problem solving, and there is hope that continued efforts may uncover more efficient methods for storing, retrieving and fixing cases [5].

Operations research approaches were conceived as static, off-line methods, and do not normally take into account real-world aspects such as incomplete data and error recovery. They require an accurate model of the agent and of the world. And they are not cheap, either. On the other hand, to our knowledge, nobody tried to work these problems out in the context of agents.

Randomizing algorithms, finally, try to do away with planning by devising experiments that "naturally" funnel into a state for which a solution (plan leading to the desired final state) is known. This approach has been used in robotics, notably in manipulation tasks, associated with clever fixtures in the end effector. It is doubtful how generally applicable this approach may be.

In many ways, planning is like programming a computer in an imperative language. The primitive actions are the programming language's various constructs. The program is a plan (sequence of instructions) that obtains the result (goal). The case based approach would consist of having a library of well known algorithms, and given a problem, automatically find and combine the appropriate algorithms. Obviously, we're not talking about something trivial. If a good, general algorithm for planning were known, computer programming would probably be radically changed from what we know today.

2.3. Adaptive agents

Adaptive agents are capable of not only planning, but also of acquiring the very knowledge required for planning – the so-called "domain knowledge". This process of "domain learning" is very different from what EBL and related schemes do. Domain learning gives the agent the ability to do things it previously was not able to. EBL and related schemes obtain performance improvements – the ability to do the same thing faster – but do not add anything the agent could not previously do. For this reason, a qualified expression such as "performance learning" should be used when referring to EBL and similar schemes; the (unfortunately common) use of "learning" in relation to EBL would suggest that EBL is capable of domain learning, which is definitely not the case.

In the context of problem solving agents, adapting means discovering and modeling the primitive actions, as well as forming the various concepts used to model the world. An attempt to do this is found in [8]. If planning is similar to automatic programming, adapting is similar to automatic programming in an unknown computer, using a language yet to be invented. Clearly, we're talking about something even less tractable than planning – if problem solving is searching in a problem space, adapting is searching in the space of problem spaces. The formalization required is something extraordinarily hard to do, and the arduous progress of science only attests to this.

On the other hand, in case-based agents, the acquisition of knowledge required for planning amounts to basically just storing new cases, which should be straightforward. The apparent contrast in the difficulty of adapting in problem solving versus case-based agents is rather startling, and should draw more attention to the case-based approach.

When is adapting important? One situation is when the environment's dynamics is continually changing, and models used in planning have to be revised accordingly. Probably, however, adapting would be more valued for obviating the process of knowledge engineering – the process of modeling of the agent and the world, normally done by a human, and known to be a notoriously difficult and tedious activity.

The case for adaptive agents, admittedly, is not as strong as for planning agents. They would, however, be very convenient, and do pose an exciting intellectual challenge.

3. The hierarchy of behaviors

In analyzing what type of behavior each type of agent is best suited for, one finds that there is no "winner". Each type is best suited for one kind of behavior and, while some behaviors are more general than others, attempts to actually reduce one to the other are invariably awkward. For example, problem solving is more general than regulation; however, attempts to implement regulation by problem solving quickly become unwieldy.

One also finds that there is a hierarchy of these behaviors in terms of the required response time and expected frequency with which each would be invoked in a typical agent, as shown in Fig. 1:

- Theory making
- Playing
- Problem Solving
- Habitual behavior
- Instinctive behavior

Fig.1 : The hierarchy of behaviors

Regulation agents implement instinctive behaviors – behaviors concerned with maintaining invariants of vital importance to the agent. In human beings, this is the function of the endocrinous system, and much of the neural system. These systems have to operate continually, and be reliable and fast.

Case-based agents implement habitual behaviors. Habitual behaviors consist basically of following patterns – procedures, routines, rituals – previously known to yield satisfaction. This type of behavior can be expected to be quite frequent in a typical agent in a stable environment, and to be relatively "effortless".

Problem solving, on the other hand, can be expected to be a lot less frequent, and require a substantially larger response time. A well known fact from psychology (and marketing) is that, once a habit has been established, problem solving (rational analysis) is very unlikely to be invoked to fulfill the same need. Also, not every habit derives from a problem previously solved. More often, habits are established by imitation, influence, persuasion, or even operational conditioning. Thus, EBL, chunking and related schemes cannot really present a general solution to habitual behavior (although the claim has been made).

Playing is a little odd in the sense that it has a very central role in the early stages of human development, but becomes relatively rare in adults. What happens here is that habitual behavior and problem solving are practically unexistent in the early stages, and playing "fills the agenda". Once the knowledge required for habitual and problem solving behaviors is obtained, playing becomes less important – it can be disastrously expensive. Playing would be implemented by randomizing agents.

Finally, theory making – the kind of stuff that an adaptive problem solving agent would do – can be expected to be very infrequent, and require a very long time to produce results.

Looking again at the hierarchy, one can see that there is a definite progression in the amount of knowledge embedded in each level, relative to the task to be done. In theory making nothing is really known and all is search. At the other end, in instinctive behavior, everything is known beforehand, and no search needs be done.

One can also see that each level *acts through the next lower level.* Theory making, for example, acts through playing (experimentation). Habitual behaviors do not directly govern end effectors; rather, they act through the controllers of the effectors – to avoid disruption, and also because there are a lot of details involved in governing an effector that have nothing to do with the agent's habits.

Each successive level does not replace the previous one, but rather uses it to extend the overall capabilities. (The same, by the way, is not true of perception: each level may require feedback that is totally irrelevant to lower levels, and thus may need to process raw input data. "Information hiding" in perception means "information loss".)

This suggests implementing agents as hierarchies of simpler agents, each specialized in the kind of behavior it implements, and with agents at one level acting through agents at the next lower level.

4. Some pitfalls and suggestions

The overall guideline in the design of agents is to embed as much knowledge as possible in the lowest possible levels of the hierarchy, and avoid, if at all possible, having to implement higher levels to achieve the required functionality. If the whole problem can be solved as a control system, excellent. If the problem can be solved structurally, with an ingenious I/O design, still better.

If planning is unavoidable, however, there are some pitfalls and suggestions we would like to point out:

4.1. Suggestion: Hierarchical, partial planning

If planning is required, let it be hierarchical and partial. Several systems have been proposed that use instead single-level, total planning – that is, the agent forms a complete plan, in terms of primitive actions, and then sets out to execute it. The problem with this approach are many:

- Where do goals come from? As seen in section 1, the concept of drive nicely answers this question, but also naturally leads to a hierarchy of goals and actions;
- Hierarchical planning is less expensive than single-level planning [14];
- A complete plan may require minute data that is not available until some initial part of the plan has already been executed;
- When an error occurs, a whole new plan has to be made – the previous (complete) plan is lost. A partial planner, in contrast, only plans enough to know what the first step should be, and lazily evaluates subsequent steps as needed. If something goes wrong, there is no big loss. A complete plan is required only at the highest level of abstraction.

4.2. Suggestion: Forward chain

Systems based on forward-chaining [16] make inferences and draw conclusions as soon as an assertion is made (that is, an event happens in the world). Systems based on backward-chaining [6, 18], on the contrary, defer inferences until there is some goal to be achieved where the new assertion may play a part.

Forward-chaining can make systems more reactive than backward-chaining, because any event can by itself initiate an immediate response from the agent, or affect what the agent is doing. The ability to interrupt a course of action and change the focus of attention comes practically for free in forward-chaining.

Similar functionality can be achieved in backward-chaining only indirectly. One way to do it is to have a higher-level planner, with the goal of monitoring any events in the world that might affect a plan in progress. If an event happens that requires immediate response, the higher-level planner would go and stop one or more lower priority plans. The situation gets more complicated if the external event changes the applicability or desirability of a plan in some subtler way. The higher-level planner should then affect the plan or its execution, but exactly what is appropriate is very application-specific. This is likely to harm the modularity of the system considerably: the higher-level planner has to know a lot about the domain, the other planners and plans in action – what events to watch for, etc. In general, it would be desirable to encapsulate this information in the lower-level planners.

Backward-chaining can also make the creation and manipulation of goals somewhat awkward. Many goals are opportunistic and come about because some object is noticed in the environment that comes in handy to satisfy some active drive. Forward-chaining models this directly. In backward-chaining, one would have to have a higher-level planner analogous to the one discussed in the previous paragraph, with similar difficulties.

Another advantage of forward-chaining in real time is that the planner can be made to examine the possible first moves from the current situation early in the planning process. Should the agent not reach a plan leading to the goal in the allowable time, it can go back and look at the possible first moves. From these, it quickly selects one that would at least leave the agent in a safer position, or gain some time (see suggestions below).

4.3. Pitfall: Complete representation of the world

Some (most?) planners assume a complete representation of the world to be maintained internally. The idea here is that there is a "perception" module that continually scans sensors and maintains an internal symbolic representation of the state of the world usable by the other modules in the system, e.g. the planner. This is a sadly inglorious proposition. The world has just too many details to ever admit a reasonably finite representation that is adequate for all purposes. The perception module can be anything but a module: it would have to intimately know the uses of the information it encapsulates. This has prompted some to proclaim, "the world is itself its best model" [1].

The painful conclusion is that, for each specific use, special perceptive processing may need to be done on the raw input data to find specific clues and events that are of no interest in other uses of the same perceptual data.

4.4. Suggestion: Use EBL or some similar scheme to avoid replanning

This is probably the least necessary of these suggestions, as most recent planners have it, but for completeness, here it goes: include some mechanism to cache or generalize solutions,

avoiding the work of replanning should the same problem have to be solved again in the future. Some working examples of how to do this are found in Prodigy [6], Soar [16] and Theo [18].

4.5. Suggestion: Have contingency reactions ready

It may happen that the agent has to react immediately, before having had the opportunity to form a plan leading to the goal. The allowable time for planning is expired, and no solution has been determined. In this situation, it is important to have ready some crude, simple contingency reaction, that at least leaves the agent in a safer position. The key here is a certain single-mindedness: the agent reacts to the immediate threat directly, without paying attention to other, possibly longer-term goals it may have, or particular details in the solution. Having put itself on safer ground, the agent can continue its planning.

Forward-chaining can determine these reflex-like reactions as soon as the event causing the urgency happens, and keep them ready if a reaction is needed without a full plan.

4.6. Suggestion: Gain time

This would be an improvement over the preceding suggestion. The idea is, in case a solution cannot be reached in the allowable time, provide some interim, revocable, cheap solution that at least gains the agent some more time to complete the plan. The central idea is that this interim solution should be innocuous and/or reversible, and take time without consuming too many resources. As the interim solution is executed, the agent can continue its planning.

Again, forward-chaining can determine these protractive alternatives early in the planning process, possibly as a sub-product of trying to find a solution that would actually lead to the goal.

4.7. Suggestion: Parallelize

There can be a bewildering amount of work to be done in an agent. Break down the work into smaller tasks. Delegate work. Parallelize aggressively. The following section discusses how this can be done.

5. Sketch of the architecture of a planning agent

Fig. 2 presents the sketch of an architecture inspired by the previous ideas:

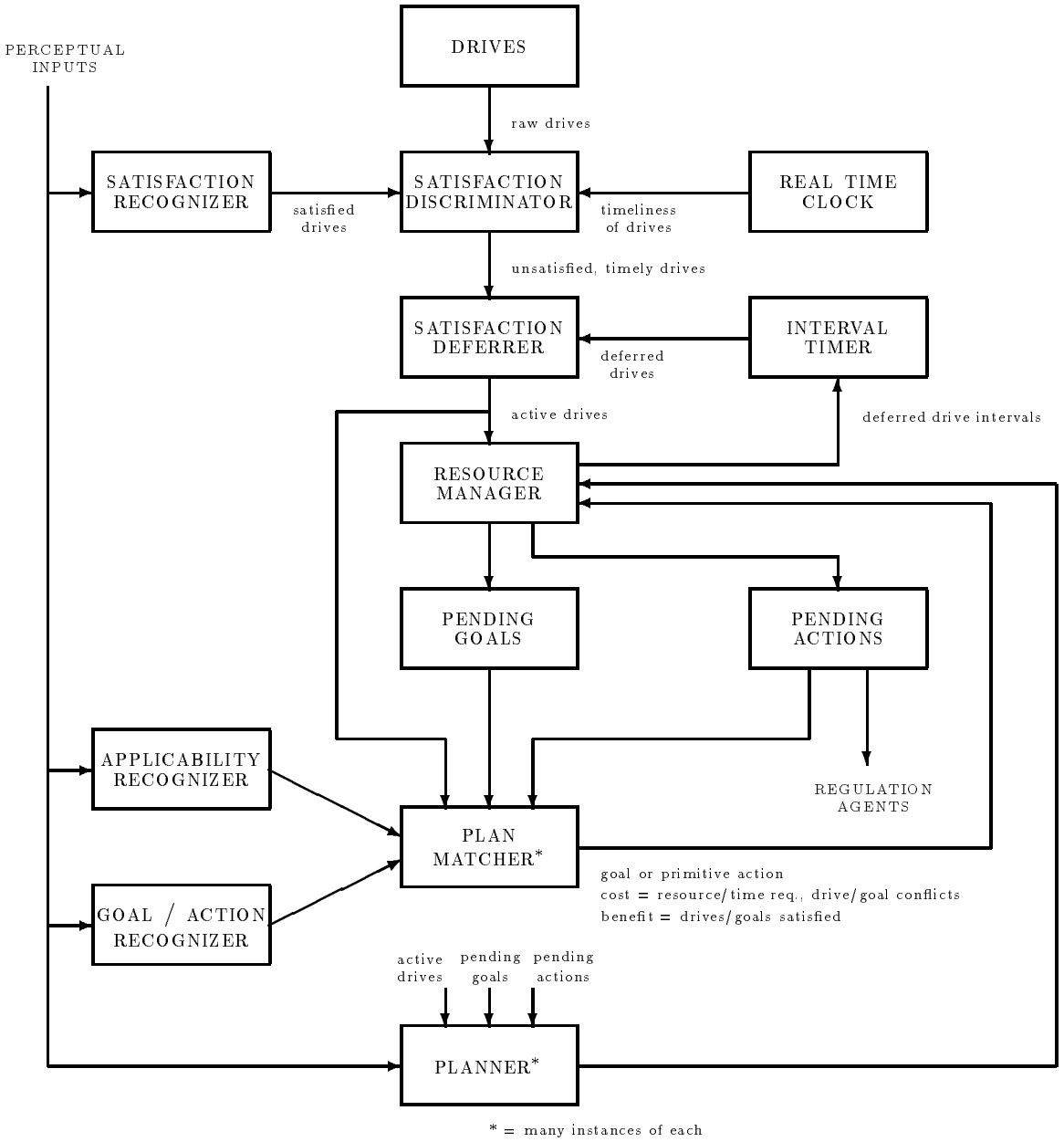

Fig.2: Architecture of a planning agent

The focal point of the architecture is the resource manager, which may be thought of as the "consciousness" or "ego" of the agent. The role of the resource manager is to decide, from among the various actions and goals proposed by the different planners and plan matchers, which ones to go ahead with.

Plan matchers may be thought of as "active memories" of plans, previously tried and known to satisfy certain drives and obtain certain goals. They form the basis for habitual behavior. A large number of matchers may be expected to be employed in a typical system, possibly each in its own processor. Each matcher continually verifies if its plan would be applicable in the current situation, and if its application would fulfill any of the active drives and/or pending goals. If so, the matcher sends to the resource manager the following information:

1. The first subsequent goal or action in the plan (the so-called "instrumental goal" or "action");
2. Estimates of the cost of this instrumental goal or action, and of the overall plan, in terms of resources and time required, as well as drive or goal conflicts (lost opportunities);
3. Estimates of the benefit of the instrumental goal or action and of the overall plan, in terms of drives satisfied or goals obtained.

If the resource manager decides to go ahead with the plan, it then posts the instrumental goal or action in the set of pending goals or pending actions, respectively. In the first case, the instrumental goal will presumably fire another plan matcher, that will further refine the goal in terms of lower-level goals or actions. In the second case, the pending action will be seen by the regulation agents, that will act accordingly on the system's effectors. In any case, the goal/action satisfaction recognizer in the plan matcher will monitor the perceptual inputs until the pending step in the plan has been fulfilled, and then signal the matcher to go ahead with the next step. The matcher then updates the information sent to the resource manager, reflecting the next step. The manager withdraws the previous instrumental goal or action from the respective pending set, and posts the new one. The cycle is then repeated.

If an instrumental goal or action does not succeed within the expected time, the matcher relinquishes the information sent to the manager, and sleeps. The manager withdraws the failed step from the respective pending set. The goal that originally fired the matcher, if any, remains pending, however. Some other plan that also matches this goal will then presumably be selected by the resource manager.

Aside from plan matchers, the system also has planners, which form the basis for problem-solving behavior. The planners continually search for plans composed of primitive actions or pre-defined plans, that would be applicable and expected to satisfy some active drive or pending goal. To do this, the planner uses the information on the applicability of each of the primitive actions and pre-defined plans, as well on the recognition of satisfaction of goals or actions (this information is the same used by the plan matchers discussed above). EBL or related schemes may be used to obtain performance improvements. Planners compete on the same basis with plan matchers, being selected if they have sufficiently favorable cost/benefit.

If more than one planner is present, each should be tuned to have a different search pattern and prioritize drives and goals differently. Otherwise, all planners would end up presenting the same plan to the resource manager.

It should be noted that, if resources exist, more than one plan may be executed at the same time. Actually, the resource manager might even decide to go ahead with more than one plan for a single goal. When the goal is achieved, the other plans would simply be dropped.

The resource manager's role is to satisfy and minimize active drives. To do so, it selects from among the various possible plans of action available, based on the expected pay-off of each, and resources and time needed. It also has to take into account possible conflicts with other drives and goals. Conflicts with drives (the so-called "reality principle") are taken care of by deferring the affected drives temporarily, until the goal may be achieved.

Not all drives are active all the time. Drives currently satisfied are not active. Also, the clock may be set up to specify the timeliness of each drive in different hours of the day or at different ages of the agent. Untimely drives are not active. Finally, as mentioned above, drives may be temporarily deferred to allow execution of a plan ("deferred gratification").

Instinctive behavior is implemented by regulation agents, which are not detailed in the figure. Any of the well-known methods of control theory and automata theory may be used in their design. The regulation agents may also reside in their own processors. A considerable amount of parallelism is allowed by the architecture.

6. Comparison with other architectures

The proposed architecture has been somewhat influenced by PRS [10, 11]. In PRS, as much knowledge as possible is embedded procedurally in "knowledge areas" (KAs), triggered by "invocation conditions". A "reasoner" selects from among the various triggered KAs. To do the selection, the reasoner invokes "meta-level KAs", that have the application-specific knowledge required to resolve which one to take. PRS can be programmed in such a way that no planning takes place. If planning is required, however, it can be accommodated. Execution is always sequential, though. At most one primitive action is executed at each time. All KAs, including meta-level KAs, are included in a single library; no modularization is attempted. No performance or domain learning is done.

KAs are similar to our plan matchers and planners, except that KAs are passive representations, whereas plan matchers and planners are active. Our resource manager is similar to the PRS reasoner, but it does not use anything like meta-level KAs. Instead, our decisions are made based on cost/benefit analysis using data provided by the matchers and planners themselves. The main idea borrowed from PRS was to provide means to avoid planning by embedding procedural knowledge of what to do in which situations.

A major distinguishing feature of our approach is the use of hierarchy and modularization to reduce complexity and achieve parallelism. The key to parallelism used here is the hierarchical decomposition of behaviors and use of "sub-agents", each with its own processor and memory. This is in contrast with PRS and other systems such as Prodigy [6] and Theo [18], where the architecture is strictly sequential. In Soar [17], parallelism is also possible, but modularity is lacking; all productions are bundled together.

A second major distinguishing feature of our architecture is that it explicitly tackles the question of how goals are formed and manipulated. Previous works have generally glossed over this issue, either by relegating it to the application, as in PRS, or by positing a "meta-level reasoner", to be implemented sometime in the future, e.g. in Prodigy.

A third distinguishing feature in relation to PRS is that our architecture can do perfor-

mance learning, in the style of EBL, in case planners are needed. PRS lacks any such ability, and cannot profit from having already solved the same problem.

7. Conclusion

Significant links exist between agents and control theory, automata and psychology. We hope that the findings of this paper will stimulate more study in these directions.

Our architecture effectively integrates different kinds of behavior into a coherent agent. We do not try to reduce all kinds of behavior into a supposedly canonical one, as done in Soar [17]. For each type of behavior – instinctive, habitual, problem-solving – we use the most efficient implementation – control systems and automata, pattern matching, planning. In contrast, Brooks' systems [3] use only control systems and automata; habitual and problem-solving behaviors are not addressed. Similarly, Theo does not accommodate habitual behavior. In general, previous architectures have focused on a particular approach, remiss to how to eventually integrate and efficiently implement other kinds of behavior that may be required.

Hierarchy plays a fundamental role in agents. Not only is hierarchy needed in planning, but also in the overall organization of behavior. As shown in the sketched architecture, this may lead to an effective partitioning and enable considerable parallelism in the implementation. It is expected that an architecture along these lines will be able to overcome the long-known performance problems in sequential implementations.

8. Acknowledgements

The author thanks Jaime Carbonell for comments on an earlier draft of the paper, and Oren Etzioni for reviewing the last version.

References

[1] Agre, P. and Chapman, D. "Pengi: An implementation of a theory of activity", in *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA, July 1987.

[2] Bernstein, A.E. and Warner, G.M. (1981) "An introduction to contemporary psychoanalysis", New York: Jason Aronson.

[3] Brooks, R. "A robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automation*, RA-2, April 1986, pp. 14-23.

[4] Cadzow, J.A. and Martens, H.R. (1970)"Discrete-time and computer control systems", Englewood Cliffs, NJ: Prentice Hall.

[5] Carbonell, J.G. "Derivational analogy: a theory of reconstructive problem solving and expertise acquisition", in R.S.Michalski, J.G.Carbonell & T.M.Mitchell (eds.), *Machine Learning: An artificial intelligence approach*. San Mateo, CA: Morgan Kaufmann.

[6] Carbonell, J.G., Knoblock, C. and Minton, S. "PRODIGY: An integrated architecture for planning and learning", in K. VanLehn (ed.) *Architectures for Intelligence*, Hillsdale, NJ: Erlbaum.

[7] Chapman, D. "Planning for conjunctive goals", in *Artificial Intelligence*, V. 32, pp. 333-377, 1987.

[8] Drescher, G.L. (in press) "Made-up minds: a constructivist approach to artificial intelligence". MIT Press, 1991.

[9] Freud,S. (1905), "Three essays on the theory of sexuality", in *Standard Edition*, Vol. 7, pp. 135-243, London: Hogarth Press.

[10] Georgeff, M.P. and Lansky, A.L. "Reactive reasoning and planning", in *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA, July 1987.

[11] Georgeff, M.P. and Ingrand, F.F. "Decision-making in an embedded reasoning system", in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989.

[12] Hammond, K. (1989). "CHEF". In C.K.Riesback & R.C.Schank (eds.), *Inside Case-Based Reasoning*, Hillsdale, NJ: Erlbaum.

[13] Knight, K. "A gentle introduction to subsymbolic computation: Connectionism for the AI researcher". Tech. Report 89-150, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1989.

[14] Knoblock, C., Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[15] Kohavi, Z. (1970) "Switching and finite automata theory", New York: McGraw Hill

[16] Laird, J.E., Rosenbloom, P.S. and Newell, A. (1986) "Chunking in Soar: The anatomy of a general learning mechanism", *Machine Learning*, 1, pp. 1-46.

[17] Laird, J.E. and Rosenbloom, P.S. (1990). "Integrating planning, execution and learning in Soar for external environments", in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Vol. 2, AAAI Press, pp. 1022-1029.

[18] Mitchell, T.M., Allen, J., Chalani, P., Cheng, J., Etzioni, O., Ringuette, M., and Schlimmer, J. "THEO: A framework for self-improving systems", in K. VanLehn (ed.) *Architectures for Intelligence*, Hillsdale, NJ: Erlbaum.

[19] Mitchell, T.M., Keller, R.M. and Kedar-Cabelli, S.T. (1986) "Explanation-based learning: A unified view", *Machine Learning*, 1, pp. 47-80.

[20] Newell, A. "Reasoning, problem solving and decision processes: The problem space as a fundamental category", Tech. Report 79-133, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, June 1979.

[21] Newell, A. and Simon, H.A. "Human Problem Solving", Prentice Hall.

[22] Ogata, K. (1970), "Modern Control Engineering", Englewood Cliffs, NJ: Prentice Hall