

Agent-based Continuous Planning

R.S.Aylett, CVE, University of Salford
A.M.Coddington, Computer Science, University of Durham
G.J.Petley, CVE, University of Salford

Abstract

The paper discusses an architecture for an agent carrying out continuous planning, in which planning and execution are interleaved and agent motivations are used to generate goals. The agent may pass actions to a number of other agents for execution. The architecture is discussed in the context of the running of a water-processing laboratory in which drinking water samples are tested.

KEYWORDS: AI Planning, Agents, Plan execution, Scheduling

1. Introduction

AI Planning technology has produced a number of powerful techniques for dealing with the sequencing of actions into a plan while coping with the vast search spaces this often produces [Penberthy & Weld 92, Weld 94], and it is increasingly being applied to real-world problems that are hard to solve with other KBS technologies [Aarup et al 94, Chien 94, Cross & Walker 94, Trave-Massuyes & Milne 97]. However the classic approach assumes that an agent can plan in a single phase, producing a plan for a particular task which it then executes in a second following phase. While there are applications which fit this model [Tate et al 96, Aylett et al 99], there are also many applications in which an agent must intertwine planning and execution.

For example, it may be that for an agent to establish the truth of the pre-conditions attached to an action it wants to add to a partial plan during the planning process, it needs sensor information which can only be acquired at execution time. This is often the case during robot planning [Aylett et al 91, Pryor 95] - for example, a robot making a plan to transport papers round an office may incorporate actions which require knowing whether doors currently out of sensor range are open or not. However, a higher-level reason for mixing planning with execution may be that an agent has a set of tasks over time which themselves must be interleaved, so that some are being executed while other are being planned. For example, the robot just discussed may be distributing the post when it notices dirty coffee cups which need to be cleared from offices to the kitchen and receives a message from the photocopier that stocks of paper are low and it needs a few extra reams delivering.

Task interleaving is typical of an autonomous agent which is situated within a particular environment and is able to generate goals itself in order to meet its overall aims and objectives. This *goal autonomy* possessed by an agent makes it rather unlike the classic AI planner which is given individual goals by a user and stops once it has achieved them. What such an agent needs is *continuous planning*, in which planning and execution are on-going activities, the agent can generate extra goals at any time, and planning is therefore never complete.

In many environments, the concept of a single agent executing all the planned actions is also unrealistic. In the work discussed below, the target environment was that of a water processing laboratory, in which drinking water samples are tested in a variety of statutorily fixed ways to confirm their quality. Here many execution agents carry out actions in parallel, so that as well as continual planning, it is also necessary to consider *indirect execution* [Myers 99] in which the agent carrying out the planning delegates

execution to many other agents. This produces a further type of interleaving, since the consideration of which agents should carry out actions - a scheduling activity - must also be incorporated into the planning process [Mussettola 94].

In this paper we discuss an architecture designed to provide continuous planning for an agent by meeting the following requirements:

1. Planning and executing are interleaved
2. The planner is able to accept new goals generated by the agent at any time
3. Time passes while the agent plans and executes, so that the planner must be able to reason about time
4. Actions are planned for many execution agents so that the planner must be able to reason about which agents should carry out which actions
5. It may not always be possible to achieve all current goals within the given time, so that it must be possible to prioritise outstanding goals. It must also be possible to remove goals which cannot be met from the planning process along with any partial plans and constraints they required.
6. The agent has long-term *motivations*, which enable it to generate goals, prioritise them, and select the best plan to achieve them

The work described has been carried out as part of the MACTA-Lab project, funded by the Engineering and Physical Sciences Research Council, and draws on earlier work carried out in the MACTA project [Aylett & Barnes 00] also funded by EPSRC. It has been implemented in Common Lisp on both a Sun SPARC station and an Intel PC running WIN98.

2. Continuous Planning

2.1 Motivations and goals

A situated agent can be thought of as an agent coupled to its environment [Maturana & Varela 87], so that changes in the environment affect its internal state and therefore its goals, and its resulting activity in turn affects the environment as well as its own internal state. Additionally, an agent that reflects - for example through planning - may take into account the predicted state of the environment as well as its current state in formulating its goals. In the architecture discussed here, these contextual effects are encapsulated in the term *motivations*, which can be seen as the root of the overall architecture in Figure 1.

Motivations can be thought of as long term aims or objectives, or as drives or emotional states [Canamero 97], depending on the domain. For example, an agent in a virtual environment might have *hunger*, *tiredness* and *curiosity* [Aylett 99] as its motivations. Motivations have an associated weight which often changes over time and provides a driving force directing the generation of goals to satisfy the motivation. As *hunger* increases, the goal of finding food will be generated, so that at some point planning activity will be stimulated to meet this goal. On the other hand, if the agent has just eaten, this motivation will not be powerful enough to generate a goal, allowing *curiosity* to generate a goal `locate-interesting-object` for example. The weight attached to the motivation is also passed on to the goals that it generates, so that a high weight attached to *hunger* will produce a high-priority goal `have-food`.

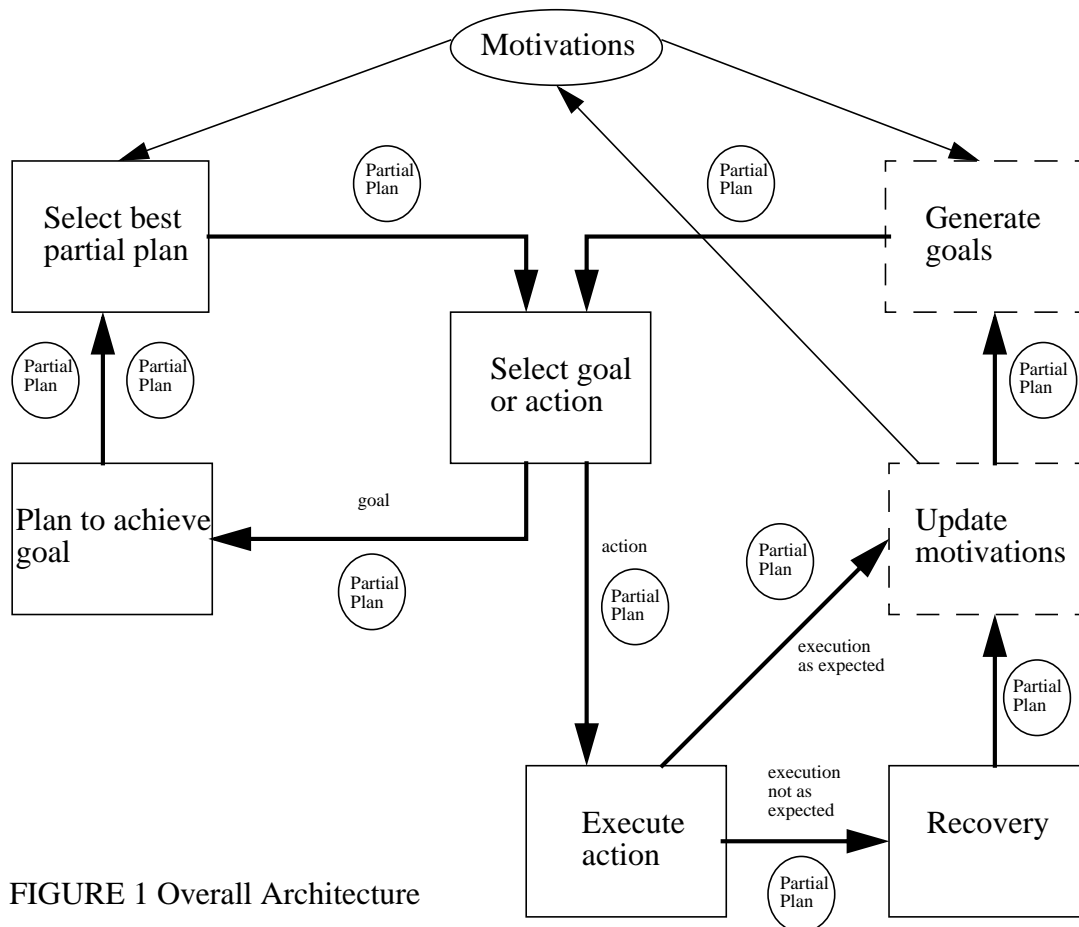


FIGURE 1 Overall Architecture

Motivations also allow an agent to evaluate the plans it generates to achieve its goals and to choose between alternative plans. For example, a plan to *have-food* that involves an agent walking to the shops to buy a sandwich may conflict with its *tiredness* motivation and result in an alternative plan involving frying an egg from a box in the fridge.

The examples given of motivations may appear less relevant to industrial domains. However, one can identify appropriate domain-specific motivations depending on the application. For example, in the case of a water-processing laboratory, one might have the motivation *deal-with-emergencies*, which would gain a high weighting if an outbreak of cryptosporidium (a small water-dwelling organism that causes illness if imbibed) takes place. This would produce a high-priority goal to test samples associated with it. One might also have the motivation *balance-load*, which would lead to the formulation of goals spreading samples out across equipment rather than overloading one particular testing station.

Motivations are represented in the current architecture by a set of tuples $\{\text{name, weight}\}$, one for each motivation. The repertoire of actions available to the agent is held in a knowledge base and indexed by a hashtable which links the name of each possible action and its parameters to the motivations that it supports or undermines. The contribution of an action to motivations is represented by two sets: the *pros* and *cons*. Each set consists of tuples $\{\text{name strength}\}$, in the case of *pros* giving the names of motivations and the degree to which the action supports them, and in the case of *cons* the motivations and the degree to which the action undermines them.

A simplifying assumption has been made in the algorithm below as it applies to the updating of motivations. This is that motivations are updated - and new goals are generated - only after an action has been executed. This reduces the computational overhead of carrying out an update with potentially far-reaching consequences, and allows the *pro* or *con* effect of the executed action to be directly fed back into the motivations to which it relates.

More realistically, one could assume that other autonomous agents in the environment might also perform actions which would impact the motivations of the planning agent. This would require it to be equipped with perceptual actions, organised by an attentional mechanism, which would search for changes in the environment that might affect its motivations. However this is left to further work, and we assume in the current architecture that the agent's own actions are the cause of changes in its motivations through their effects on the environment.

2.2 Representations

The previous section discussed motivations; here we outline the way in which the actions and goals involved in planning and execution have been represented.

Table 1: Action Format

id	a unique identifier
type	specifies whether an action or an event is represented
name	gives the name of the action and its parameters: e.g. movebottle (?bo, ?fm, ?to)
agents	the set of agents required for execution; e.g. (human ?h) or (robot ?r)
precond	which facts must be true for the action to be executed
add	facts which become true as a result of executing the action
delete	facts true before execution but no longer true after execution
pros	a set of {name strength} tuples for motivations supported by action
cons	a set of {name strength} tuples for motivations undermined by action
goals	a set of goal IDs for goals generated by goal generator to which this action contributes

While in many respects a standard representation going all the way back to the STRIPS planner of the 1970s, actions include the contribution to the agent's motivations discussed above. An action also specifies which agent - or agents - are required to execute it, in order to support the allocation of agents to actions for indirect execution, also mentioned above. Finally, the `goals` field enables the planner to keep track of dependencies, so that goals and all the actions which support them may be removed from the plan,

Associated with each action as it is inserted into a partial plan is a further structure called a *node*.

id	node identifier: same as the ID of the action or goal it is associated with
type	:action, :event, or :goal depending on which the node is associated with
importance	of the node's corresponding action or goal (low = more important)
effort	amount of work expended (see discussion below)
window	earliest and latest start times
duration	how long associated goal or action takes

Table 2 - Node format

Nodes can be associated with goals (and with events) as well as with actions, and record information used both in planning and in execution. In particular the `effort` field indicates the cumulative effort expended in achieving a main overall goal or goals. If the node is associated with such a goal, then the `effort` indicates the amount of work involved in selecting the actions and constraints which, once executed, meet that goal. If the node corresponds to a subgoal, an action, or an event, the `effort` indicates the amount of work expended in achieving the goal to which the subgoal, action or event contributes.

The `window` and `duration` fields are used to reason about time. Again, their meaning depends on whether the node is associated with an action or with a goal. If an action, then the `window` indicates the time range in which the action must start executing, and the `duration` how long it is expected to take to execute. If a goal, then the `window` refers to the time range in which the goal must be achieved, and the `duration` to the length of time it must remain true.

Goals may be produced in this architecture in three ways. Firstly, the goal generator may create goals in response to the agent's motivations, as discussed above. Secondly, a goal may be generated as a subgoal by the planning process itself - when an action is inserted into a partial plan in order to achieve some desired active goal, its preconditions are added as subgoals to be achieved, or *open conditions*. Finally, the recovery component of the architecture in Figure 1 generates goals or subgoals which are to be re-achieved when executing an action does not have the desired effects.

id	goal identifier
type	:goal or :subgoal
condition	a predicate representing the goal or open condition
importance	of the goal
effort	amount of work needed to achieve a main goal
window	earliest and latest times for goal achievement
duration	how long goal must remain true

Table 3 - Goal representation

In Table 3, the goals added as open conditions during planning only have the first three fields in representation shown. Because they are derived from an action's preconditions, the node associated with the action is inherited by the goal and contains the information in the latter four fields of Table 3. In addition, the ID of an open condition is the same as the action from which it is derived, allowing such goals to be tied to the action which produced them.

2.3 The main algorithm

The high-level algorithm for the planning system is as follows (read in conjunction with Figure 1) and will be discussed below:

```
while agent active
do
```

```

    Select best partial plan
    Select between goal or action to execute
    If goal selected
    then
        generate new partial plan
    else
        /* executing action */
        execute action
        determine outcome
        update plan
        update motivations
        generate goals
        retract old partial plans
        update clock
    endif
endwhile

```

Choosing the best partial plan simply involves taking the plan at the head of the search space of partial plans, which is composed of a set of (plan, value) tuples ordered on the basis of value. Values are calculated in relation to the motivations supported by the actions in the partial plan.

In order to select between planning to meet a goal or executing an action, it is necessary to pick the best goal from the set of goals to be achieved and to determine which actions are currently ready to be executed. Goals are evaluated according to the importance, effort and deadline associated with each. High importance is self-explanatory. High effort suggests that the agent has already invested in the achievement of the goal and may suggest that the goal is some way to being achieved. For both these reasons the goal should score more highly in the selection process. Finally, goals with an imminent deadline should be preferred to those with a slacker time constraint.

An action is ready to execute if it is first or joint first in a partial plan, if all its preconditions are true in the current state and if the current time falls within its execution window. Its value is calculated in the same way as for a goal - by summing its importance, effort and deadline fields. Thus it is straightforward to compare the best goal and the best action ready for execution in order to determine whether to plan or to execute on the current cycle.

Planning takes place if a goal was selected, and a new partial plan is added to the set currently available for execution unless it proves impossible to meet the goal at all. When the plan has been produced, deadlines are estimated for all of its actions, based on the start/end times and durations recorded there. The approach is based on that of [Vere 83]. The plan is then checked to see if there is enough time to meet all of its goals. If not, the current system offers the user the opportunity to edit the plan to change the time constraints, and if this is not taken fails the plan and removes the goal it was trying to meet from the active set. Otherwise the goal is removed from the active set (as the plan will meet it when it executes) and the plan is added to those under consideration.

2.3.1 Executing an action.

Execution takes place if an action was selected rather than a goal. The action is dispatched to the execution agent, which in the current system involves sending a message down a socket to a discrete event simulator (DES) holding a representation of the water processing laboratory. The DES responds with a message stating the time and the state of the world after execution. In other environments, detecting the state of

the world after execution might involve sensing actions by the agent, or in domains where execution is widely delegated, messages from individual execution agents reporting their sensing of the resulting state of the world. In order to support debugging, there is also a facility to allow these values to be entered manually.

If execution does not return the expected state - the outcomes held in the `add` and `delete` list of the action - then the state returned is examined to see if it violates links in the plan to which the action belongs. If so, then these links are converted into goals - they represent goals which should have been achieved but now have to be achieved in some other way. These goals are added back to the partial plan to which the action belonged, which is also updated to remove the executed action. Thus replanning is an integral part of the planning and execution process since these goals will require extra planning activity and the subsequent actions that depended on them will not be executable until this has been done.

Following execution, the agent's motivations are updated to reflect the changes in the world. If the action has been successfully executed, the `pros` and `cons` fields can be used to perform this update, though as suggested above, a more sophisticated scheme would involve perceptual actions by the agent. How - or whether - motivations should be updated if an action fails has not yet been investigated. Again, a facility to enter new motivation values manually has been provided for debugging.

Next, new goals are generated. In the water processing lab, a new goal is provided by the arrival of more samples to process, which can be modelled as a stochastic event in the DES. In a more complex scenario, one might have machine maintenance or repair events also. Having generated new goals, all previous partial plans in the search space are thrown away as the world model they assumed as their initial state may no longer correspond to the world after the execution of the action. This is of course a simplifying assumption: it would be better to check the changes in the world against that assumed by each plan and only jettison those plans for which the start state really has been violated, but this complexity remains to be tackled.

Finally, in the current version, the clock is updated to reflect the amount of time taken to execute the action. A future version supporting asynchronous interaction between the components of the architecture might lead to modification of this part of the algorithm.

3. The Water Processing Laboratory

Figure 2 gives an overview of the activity of a water processing laboratory, which is carried out under the supervision of a Laboratory Information Management System, or LIMS, a conventional software package organised around a database. The continuous planning system discussed above can be thought of as an intelligent LIMS.

The process is driven by a customer request, which in the case of drinking water samples would normally be a water supply company. Indeed, water processing laboratories were originally an integral part of water authorities, but privatisation has led to the creation of independent laboratories. Many still derive the majority of their business from the water supply company with which they were originally associated, but are diversifying fairly rapidly. Many customer requests are routine - there is a statutory daily regime for the testing of drinking water - but one-offs and emergency requests are becoming more common

A customer request normally leads to the dispatch of containers for sample collection in the field – these may contain specialised preservative chemicals in some cases. Thus testing can be planned from the time the request comes in, but with deadlines usually at least a day in the future. This might not be the case if there was an emergency request however – for example suspected cryptosporidium contamination, as already mentioned, or disasters such as the aluminium sulphate contamination in Camelford in the 1990s. Samples arrive in multiple bottles, depending on the number of tests to be carried out; up to 23 have been noted during investigations.

Sample preparation is carried out in the individual sections within the laboratory, and depends on the test being carried out. For example, if testing for organic contaminant, a solvent such as pentane may be used in order to extract the contaminant. The pentane is then evaporated off until only the contaminant remains. Alternatively, the sample is sucked through a tube (using vacuum pumps) containing solid phase powder which removes the contaminant. A solvent is used to extract the contaminant from the powder.

Sample testing involves a variety of equipment and processes depending on the test. Many tests require a batch of fixed size – say 20 samples – so that number has to go through sample preparation before the batch can be tested. Samples may be tested for a wide range of organic contaminants, for metals contamination, and for physical characteristics such as pH, true colour, turbidity and conductivity. In some cases sample handling is carried out by human staff, and in others, where testing is automated, by robots. The results of tests are collected, and dispatched back to the customers, with email and the web playing an increasing role.

It is this process which is being modelled in the Discrete Event Simulator, which serves both as an execution system and as a model of the real-world for the continuous planning system. The DES can be used to implement scenarios with variable customer orders, variable execution times, and unexpected events such as machine outages. The agent's planning system is coupled to the DES using sockets, and the execution agents are represented by the humans and machines defined in the DES scenarios.

4. Conclusions and further work

We have discussed here the representations and high-level algorithm needed to give an agent the ability to carry out continual planning. The water processing laboratory represents an interesting test bed for this type of planning since by definition its work is never finished and planning and execution are naturally interleaved. To the extent that the LIMS drives the laboratory, the concept of indirect execution is already applied, though the LIMS has no ability to deal with anything other than the expected sequence of events. Generative planning, if combined with time management and resource allocation as discussed above, is capable of dealing far more flexibly with the variation of work load that is increasingly becoming the rule as laboratories develop more diverse customer sets and look for new services to offer.

However the system presented above, though developed for a laboratory domain, is generic and capable of much wider application. It gives an agent the ability to combine reflection with reaction and planning with acting, as well as to delegate execution activity to other agents. The concept of motivations provides a potential link between symbolic reflective abilities and a non-symbolic system running at a lower level within an agent which could be applied to a robot agent or to an agent operating in a virtual environment [Aylett 99].

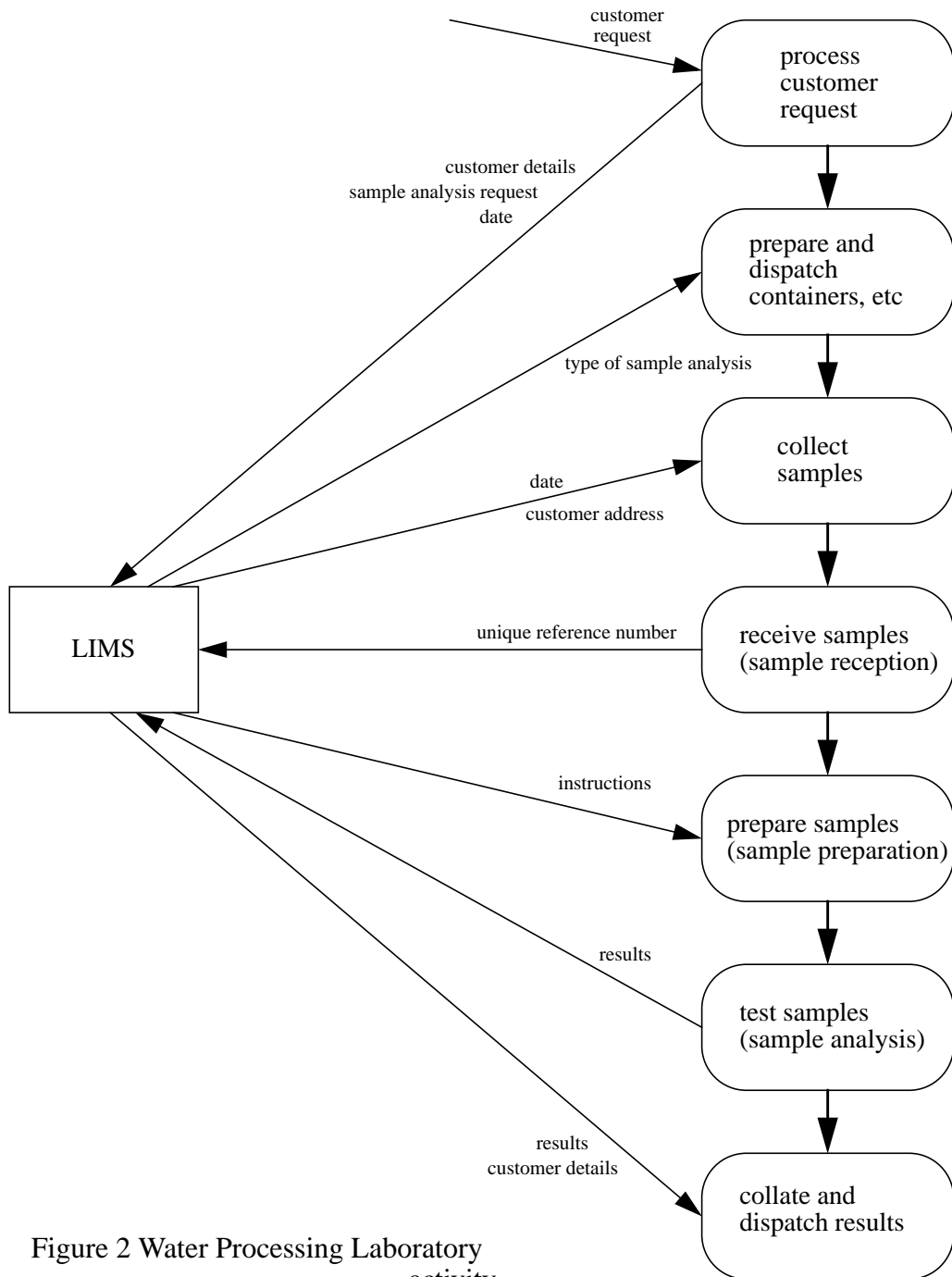


Figure 2 Water Processing Laboratory activity

References

- Aarup, M; Arentoft, M.M; Parrod, Y; Stokes, I; Vadon, H. & Stager, J. (1994) "Optimum-AIV: A Knowledge-Based Planning and Scheduling System for Spacecraft AIV". In Intelligent Scheduling, M.Zweben & M.S.Fox (eds) Morgan Kaufmann 1994, pp451-469
- Aylett, R.S; Fish, A.N; Bartrum, S.R (1991) HELP: A Hierarchical Execution-led Planner for Robotic Domains, Proceedings, ESWP-91, Springer-Verlag
- Aylett, R.S; Petley, G.J; Chung, P.W.H; Soutter, J. & Rushton, A. (1999) "Generating Operating

Procedures for Chemical Process Plants” Integrated Manufacturing Systems - The International Journal of Manufacturing Technology Management 10(6) pp 328-42
1999 ISSN 0957-6061

- Aylett, R.S (1999) “Behavioural Virtual Agents”. In: M.Wooldridge, M.Veleso (Eds) AI Today, Springer-Verlag Lecture Notes in Artificial Intelligence, no1600, pp1-12, ISBN 3-540-66428-9, 1999
- Aylett, R.S. & Barnes, D.P. (2000) “Connecting reflection and reaction - a heterogeneous multi-agent architecture”. In: Human Cognition and Social Agent technology, ed. K.Dautenhahn, pp 197-224 in series “Advances in Consciousness research”, ed M.Stamenov & G.Globus, John Benjamins Publishing co. Oct 99 ISBN 9027251398
- Canamero, D. (1987) Modeling Motivations and Emotions as a Basis for Intelligent Behaviour. In W.Lewis Johnson (ed) Proceedings of the 1st International Conference on Autonomous Agents, ACM press, pp148-55
- Chien, S. (1994) "Using AI Planning Techniques to Automatically Generate Image Processing Procedures: A Preliminary Report" Proc. AIPS94, Chicago, IL, June 1994, pp219-224
- Coddington, A.M. (1998) Self-motivated planning in autonomous agents. 17th Workshop of the UK Planning SIG pp49-59
- Coddington, A.M. (2000) Self-motivated planning in autonomous agents. PhD thesis, University of London, 2000
- Cross, S.E & Walker, E. (1994) "DART: applying knowledge-based planning and scheduling to crisis action planning". In: Intelligent Scheduling, M.Zweben & M.S.Fox (eds) Morgan Kaufmann 1994
- Dean, T; Firby, J. & Miller, D. (1987) Hierarchical planning involving deadlines, travel time and resources. Computational Intelligence vol 4 no 4 pp381-398 1987
- Maturana, H. & Varela, F. (1987) The Tree of Knowledge. Boston MA New Scienc Library
- Muscettola, N. (1994) HSTS: Integrating Planning and Scheduling. In: Intelligent Scheduling, ed M.Zweben & M.S.Fox, pp169-212, Morgan Kaufmann, 1994
- Myers, K.L (1999) CPEF: A Continuous Planning and Execution Framework. AI Magazine, vol 20 no 4, pp63-69
- Penberthy, J.S. & Weld, D.S. (1992) ‘UCPOP: A Sound, Complete, Partial-order Planner for ADL’. Proceedings, KR-92, pp 324-32
- Pryor, L. (1995) "Decisions, Decisions: Knowledge Goals in Planning". In: Hybrid problems, hybrid solutions (Proceedings of AISB 95) J.Hallam, ed, pp181-192 IOS Press 1995
- Tate, A; Drabble, B. and Dalton, J. (1996) O-Plan: A Knowledge-Based Planner and its Application to Logistics"; , AIAI; In Advanced Planning Technology, The Technological Achievements of the ARPA/Rome Laboratory Planning Initiative; (ed. Tate, A.), AAAI Press; Menlo Park, California; May 1996;
- Trave-Massuyes, L. & Milne, R. (1997) "Gas-Turbine Condition Monitoring Using Qualitative Model-Based Diagnosis" IEEE Expert: Intelligent Systems and their Applications, IEEE Computer Society. May/June 1997 pp22-31
- Vere, S.A. (1983) Planning in Time: Windows and Durations for Activities and Goals. Pattern Analysis and Machine Intelligence, vol 5, pp246-267, IEEE1983