A Computational Model for Multiple Goals

Stathis Kasderidis

Foundation for Research and Technology - Hellas
Institute of Computer Science, Computational Vision and Robotics Laboratory
Science and Technology Park of Crete
Vassilika Vouton, P.O. Box 1385, 71110 Heraklion, Greece
stathis@ics.forth.gr

Abstract. The paper discusses a computational model suitable for the monitoring and execution of multiple co-existing goals inside an autonomous agent. The model uses a number of mechanisms to calculate dynamically goal priority. We provide an overview of the model, a discussion of a Cognitive Agent architecture that includes it and we provide results that support the current design. We conclude with a discussion of the results, points of interest and future work.

1 Introduction

There has been in the past decade great interest and advancements for planning algorithms in the AI literature [1] (and references therein). The main focus of such algorithms is the development of a suitable plan that will achieve a target state. For autonomous agents, with arbitrary long lifetimes, situated in a dynamic environment the above approach was extended with execution monitoring and re-planning facilities. Usually the framing of the problem took the form of Hierarchical Task Networks [1], or that of Abstraction and Hierarchical Planning [2, 3].

However, while the above approaches clearly attack an important aspect of the overall problem, they do not attack the full problem. For this reason we suggest that the whole planning problem be decomposed into distinct but interwoven subproblems: i. The problem of 'reasoning', ii. The problem of 'execution'. The first problem deals mainly with the development of a suitable plan which will achieve a target state, while the second one is related to the monitoring of the plan and the provision of re-planning requests to the 'reasoning' component dynamically.

This paper will discuss the 'execution' component of the planning process inside the agent. The paper is structured as follows: in section 2 we will provide a concise overview of the Cognitive Architecture and of the links of the execution component (which we call the Computational Model) with the Reasoning, Motivation, Concept System and Goal Generation modules of the architecture. In section 3 we present the Computational Model and justify our motivation. In section 4 we provide results that support the current work. Section 5 concludes with a discussion on various points of interest and future work.

2 A Cognitive Architecture

The GNOSYS Architecture aims to control robotic agents in arbitrary complex, novel and unstructured environments. It consists of a number of high-level modules that provide the basic cognitive capabilities of the agent. For the purposes of this paper, figure 1 provides a reduced version, including modules only necessary for the discussion of our Computational Model.

The information flow starts with the request for the achievement of a target state. This request comes from two primary sources. It is either a *User* request or a request from the *Motivational System*. The realisation of the request takes the form of *Goal* object, which is generated by the *Goal Generation System.* The Goal includes a plan and a number of parameters (for definition see section 2.1). The necessary plan is provided by the Reasoning System, while initial values of the parameters are given by the Concept System and corresponding Value Maps. After the Goal object is created and appropriately parameterised inside the Goal Generation module it is sent to the Goal Space. There it becomes active and starts its life as a self-contained execution entity. In effect the Goal Space corresponds to the notion of the ‘core execution loop’ for our architecture.

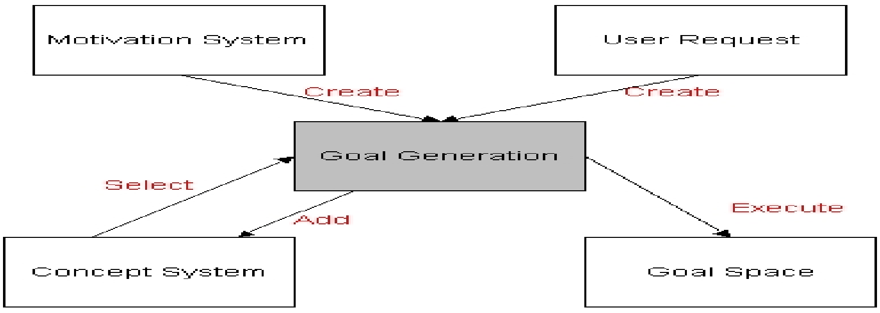

Fig. 1. Components of the GNOSYS Architecture and relations to Goal Space (which implements the Computational Model)

The Goal object and the Goal Space constitute our structural decomposition of the Computational Model while the functional decomposition corresponds to the actual mechanisms that control the lifetime and priority of a given goal; they will be discussed in sections 2 and 3 respectively.

2.1 Definition of a Goal

A Goal object is a self-contained entity (an execution scope). It encapsulates the following components: A goal state; A plan which will provide the sequence of actions needed in order to achieve the goal state; Various parameters that control the lifetime and priority of the Goal. This type of encapsulation might seem strange at first but the justification behind this definition follows a number of ideas:

i. Partition of Output Space in disjoint action domains;
ii. To keep statistics about the utility of a plan for reaching a target state
iii. To allow for the case of Hierarchical Plans for the Reasoning System
iv. To allow the representation of a Goal as a concept.

2.1.1 Partition of the Output Space

The *Output Space* is the space of actions produced by the agent. These actions might be directed to the agent's environment or internally to the agent itself. The main idea for handling and monitoring the complexity of generated action is to use a divide-and-conquer strategy. This leads naturally to the division of the output space in mutually exclusive domains of 'action'. Goals in a given domain compete with a WTA strategy for the right of realising their actions in the current processing step. They are organised as a set of competing families. Each family represents a plan with a specific structure. This will be discussed further in section 3.2. This approach allows the overall architecture to scale efficiently with the dimensionality of the problem and to accept distributed implementation so as to bring in more computational resources as needed.

2.1.2 Goals and Reinforcement Learning

So far we have not explicitly mentioned any Learning process that takes place inside our agent. Indeed such functionality is present. The Value Maps module stores maps that associate states with actions using a RL learning paradigm. However, while the RL framework is very clear on how to build these associations when one uses primitive actions and states, complications arise when we have 'abstract actions' (and abstract states), where we use a hierarchical planner to decompose a high-level task to finer and finer plans up to the level of primitive actions. Care is needed when abstract actions and states are involved. This is due to the context that initiated the related primitive actions. The same primitive action can be evoked by a number of more abstract plans, so the action should not be treated only as a simple action but as primitive Goal; in this way it can be linked to information regarding the actual hierarchical plan that is member of. The standard RL framework can be extended in this way to include the effects of internal context (created by the processing of multiple co-existing Goals) The utility of a plan (schema) depends not only on the sum of utilities of its primitive actions, but also on the contextual information related to the utilities of other co-executing plans.

2.1.3 Reasoning and Hierarchical Plans

While it is not the purpose of this paper to discuss the actual problem of development of a Reasoning system, we have hinted so far that such a system is based partially in a hierarchical planner. Our primary concern is to treat a plan that as a whole that must compete and evaluate its priority. For this reason it is more natural, to our opinion, to call the plan and its goal state a Goal and then take the view that a Reasoning system needs to define a 'meta-plan' (a *decomposition pattern*) that decomposes a high-level goal into a sequence of sub-goals. Having said that we do not want to be more specific as to the nature of the 'meta-plan'. A brief discussion will be given in section 3.2. Independent of the form of the actual meta-plan we can design mechanisms that are mostly independent of the decomposition pattern used and are based on general ideas.

2.1.4 Goals as Concepts

In the GNOSYS Architecture all internal representations are treated as concepts at a suitable level of abstraction. For example Object, Schema & Belief concepts exists. A schema is an abstract plan that is parameterised with appropriate concrete target states and a set of allowable primitive actions, constraints, etc. These aspects constitute features of the concept, which take specific values with a given instantiation. To treat schemas as concepts the Concept System represents them as feature sets and creates links from one schema to another due to shared sub-plans, value maps, or other such features. Thus one can easily imagine semantic-style networks of concepts that relate Goals with each other in the same or different abstraction levels. These semantic networks are built through agent experience and are updated dynamically. The end result is that when we search appropriate initial values for the parameterisation of a Goal (by the Goal Generation System), we can retrieve appropriate information by the use of stored information in the Concept System.

2.1.5 Goal Attributes

Attributes of a Goal can be separated into a number of classes according to their use. A high-level classification is as follows:

- Identification Flags (includes ID, Parent ID, Goal class (i.e. concept), Owner, Domain ID)
- Type Flags (includes Mode and Type (Primitive or Complex))
- Importance Flags (includes Weight, Emotional Weight)
- Action-Attention Flags (Action Index, Sensory Attention Index, Motor Attention index, Boundary Attention Index)
- Drive Flags (includes Commitment, Engagement)
- Termination Flags (Termination Probability, Elimination Probability, Execution Count, User Termination Flag)
- Resource Flags (includes the IDs of the corresponding action domains that the Goal executes in as well as IDs of input modalities)

Most of the above classes are self-explanatory. It suffices to say that Identification Flags cover also the Goal class (as a concept), the Owner (which could be the 'Self' concept or the agent's user) and the Domain ID. The latter is used to denote the domain that the Goal competes into. A Goal can exist and compete independently into multiple domains. For example a high-level Goal of 'fetch the cup from the table' can be translated into a plan of moving from an initial point to a point closer to the table (one action domain – BodyMove) and start a combined sub-plan of movement and grasping actions such as to position the robot body in a suitable pose so as the robotic Arm to grasp the target object. In this sub-plan we have the appearance of a second action domain (that of the Robotic Arm). In the case of the sub-plan the actual high-level command of Fetch (CupOnTheTable), exists concurrently in both domains and acts as a family leader that spawns children sub-goals. For the BodyMove domain these might be Goals such as Move, Rotate, etc, and for the Arm domain these might be Goals such as MoveArm, Grasp, Open/Close Fingers, etc. The importance, drive, action-attention and termination flags are used by the various mechanisms that constitute the functional decomposition of the Computational Model. They will be discussed in section 3. Finally the Resource Flags provide the IDs of the Action Domains that the Goal will be registered with to compete into and the IDs of the input modalities. The notion of the input modalities corresponds to the partition of the input space (an analogous case to the Output space discussed above) into separate and mutually exclusive sources of input information. The set of input modalities provide the State Vector of the agent in a given time. The input modalities include both environmental and internal sources of information.

2.1.6 Goal Structure

Figure 2 shows the internal structure of goal as a set of modules. Each module corresponds to a sub-process that executes in the scope of the Goal.

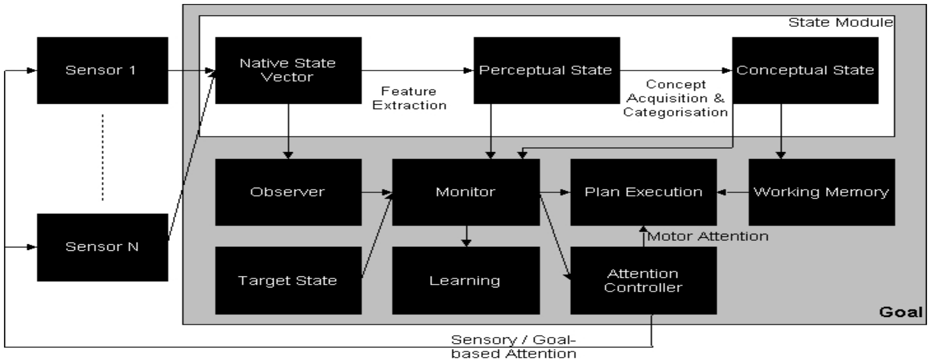

Fig. 2. Components of a Goal Object. See text for description.

Sensors correspond to input modalities. The State Module transforms the native (raw signals) state into perceptual representations (i.e. set of features). The perceptual state is transformed to a conceptual one by recognition of present object concepts in the sensory input. If novel objects exist new concepts are formed in the concept system. In either case, instances of the concepts are activated in the working memory module. This information is used next by the plan execution module. The plan is provided by the reasoning system and it is executed by the plan execution component. It is the module that implements the selected action in the current processing step. After an action is executed a new cycle begins. Erroneous plan execution is indicated by motor attention events. The target state module provides a representation of the goal state. The observer module builds a model of the environment / internal state as needed. The monitor module provides a comparison of the current state with the target state so as to signal the termination of the plan. It also measures discrepancies between the expected state (coming from the observer) and the actual state so as to raise sensory attention events. The attention events include sensory, motor and boundary attention events. These are stored in the attention controller module, which implements a conceptual queue. Attention events might request re-planning through the reasoning system. Part of the plan may be the change of the resolution of the state information through the input modalities (this aims to either enhance information about particular objects, *goal-based* attention, or to better sample novel signals, *sensory attention*). If there are discrepancies in the prediction of the observer the monitor module initiates an on-line learning phase of the current observer models. The learning module implements an RL mechanism, which learns utilities of state/action pairs. These are stored in value maps in the concept system. At the termination of the Goal the various models are stored back to the concept system as features of the current Goal and are associated with the Goal's parents as an indication of the computational context that produced them.

The internal operation of a Goal is shown schematically in figure 2. A Goal object terminates in two ways:

i. By achieving its Target State;
ii. By elimination from the Goal Space.

The actual definition of reaching the target state is included in the Monitor module of fig. 2 and obviously depends on the state representation. In our case the states are considered to live in continuous spaces and we use a Euclidean distance between the current state and the target one. In practice we compare this distance with another attribute (not described in section 2.1.5) to determine if the target state has been approached within an ε-neighbourhood.

2.2 Definition of the Goal Space

The Goal Space acts as the highest-level container of the Goal objects in our architecture. It is divided in a number of action-domains that correspond to the actual output modalities. However the active number of action domains at any given time depends on the classes of Goals that are currently executed. Figure 3 provides a schematic representation of the Goal Space of our agent and the legend provides an explanation of the concept.

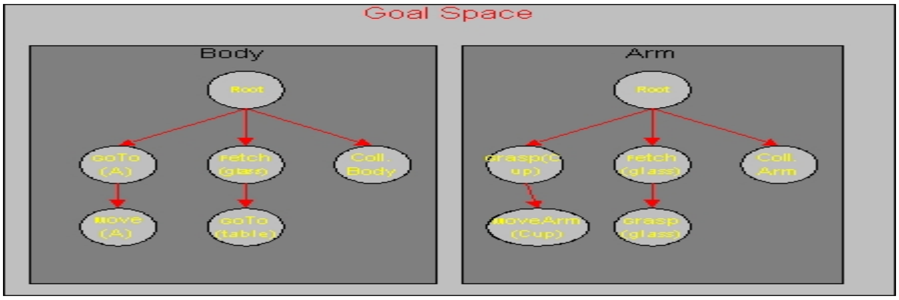

Fig. 3. Goal Space of the GNOSYS Agent architecture. Two action domains are currently active. In both domains three family trees compete for access to actual output. This is called *Global Attention Control* and the winning family will produce actual results. The outputs from both domains are executed in parallel as they represent mutually exclusive sub-sets of the action space.

2.2.1 Action Domains

The concept of the Action Domain has already been explained above. It suffices to add that the actual grouping of the competing Goals takes place as a tree of Goals (see fig. 3), which we call GoalTree. The actual process of calculating the priorities is by the use of the *Action Index* concept. It will be discussed in section 3 more fully. It is a real-valued variable bounded in the interval [0,1]. The actual process of priority calculation has the following high-level steps: Termination Check; Process; Action Index Calculation; Elimination; Output.

The first step checks the termination conditions of a Goal object and if they are satisfied, it is deleted from the current GoalTree. The remaining Goals (if any) continue to the Process phase where they calculate their responses for the current state input. Their responses are added in a list of virtual actions that is maintained by the domain. In the next step the Action Index Calculation takes place in a recursive manner from the Family Parent to its children and so forth. In the fourth step, the winning family is identified and the suggested virtual actions of the losing families are eliminated from the virtual actions list. The determination of the winning family is based on the value of the highest action index. Finally the virtual actions are sent to the actuators to be realised as output. Another processing step starts again. Steps should be taken so as to terminate Goal families that do not seem to converge to their target state. This issue will be discussed next in section 3.

3 A Computational Model

In this section we provide a description of the actual mechanisms that underlie our computational model. Two distinct issues must be discussed. The first one relates to the calculation of the goal priority and termination conditions. The second one relates to the Goal/sub-Goal dependencies and how these affect the computation of scheduling priorities. These issues are discussed next in their corresponding sections.

3.1 Goal Priority and Termination

A Goal's priority is effectively given by the value of its Action Index. The actual formula is given by (1):

$$\text{Action Index}=(W + EW + \text{TermProb} + \text{S-AI} + \text{M-AI} + \text{B-AI} + \textstyle\sum_j \delta(j, \text{contribut.})) \,/\, (6+\#\ \text{Contributing Children}) \quad (1)$$

Let us explain the main components of formula (1). We assume that each Goal has an *extrinsic value* (W) that comes either by the User or by experience (built by RL and it is thus stored in the value maps). This weight, as all other terms in (1), is suitably scaled in the interval [0, 1]. The term EW represents the *intrinsic value* of the Goal. This value is updated in every processing step by the Motivation System and captures the influence of the Goal to the agent's well being. This idea allows us to assign different value on a Goal even when the agent faces the same external state. In this case the value of a Goal will depend critically on the importance that the Motivation places on the Goal in comparison with other co-existing Goals. The Termination Probability is discussed below. The terms S-AI, M-AI and B-AI correspond to three different (local) attention mechanisms that capture complimentary aspects. S-AI corresponds to *sensory attention*, which captures novelty in the environment. M-AI captures related *motor attention* events. This idea roughly corresponds to the fact that in a real system, actions might not be performed perfectly, or that they will not terminate in the expected time. A motor attention event does not necessarily imply a need for re-planning; it might simply indicate the need for a second attempt to achieve the current plan action. However after a number of unsuccessful re-trials this will indicate a need for re-planning. The B-AI captures attention events coming from the internal agent environment; more specifically the deviation of one or more *homeostatic variables* from their equilibrium state. It is called *boundary attention*. Suitable definitions and concrete example definitions of these concepts can be found in [4, 5]. The last contribution comes from a Goal's children and corresponds to the idea that an attention event, which might be raised in any level of a Goal hierarchy, might influence the execution of the high-level Goals as well. For this reason it is prudent to raise the whole family's priority. The actual mechanism in (1) uses the number of *contributing children* for a given goal. A goal is called contributing if it produces an attention event or it has a child that produces such an event.

Normally a Goal is terminated if it converges to its target state, if it is executed a set number of times, or it is stopped by the User. However there is the pathological case where Goals might not converge to their targets, due to environment changes, interference from other Goals' plans or otherwise. In such case the actual Termination Check in 2.2.1 evaluates first the *formal termination conditions* for a Goal. If these are not satisfied, it generates a random number in the range [0, Elimination Probability] and compares this against the Termination Probability. If the Termination Probability is less than the random number, the Goal is eliminated from the GoalTree. In our current implementation the Termination Probability is a function of the distances between the current and the target states when sampled in appropriate intervals, e.g. every 5 secs. The Elimination Prob. is a more complex concept. In every processing step a Goal receives from the Motivational System updates for the *Commitment* and *Engagement* variables. Commitment in our architecture captures the persistence that the agent places to the Goal. Engagement captures higher-order effects that indicate agent withdrawal due to a prolonged period of low emotional importance (EW) or of suspended execution. Thus it is a function of EW and Termination Probability. We do not provide here formulas for Commitment and Engagement as these are actually components of the Motivational System and will be discussed there in a different paper. For the current paper it suffices to say that the Elimination Probability has been defined as in (2).

$$\text{ElimProb}=1.0-\text{Commitment}*\text{Engagement} \tag{2}$$

During Goal initialisation the corresponding values of Commitment and Engagement are set to 0.9. They are both bounded in [0, 1].

3.2 Goals, Plans and Priorities

So far we have mentioned that the Reasoning System uses partially a hierarchical / abstraction approach to planning. Without actually going to the details of how this might be achieved it is useful to identify the decomposition classes that might exist in our architecture. These include the following:

A. Independent sub-Goals (a. Sequential steps, b. Concurrent Cooperative, c. Concurrent Competitive)
B. Dependent sub-Goals (a. Serialisable, b. Block Serialisable, c. High-order dependence)

Let us offer some examples of the above patterns. Case A.a corresponds to a set of movement commands to a robot to visit points of interest in a room. Case A.b corresponds to the case of concurrent robot movements to achieve a pose against a table in order to lift an occluded object, while at the same time moves its robotic arm. Case A.c corresponds to an example of movement where we take care of collision avoidance with other objects in the environment. Serialisable Goals are those which a specific sequence of commands must be carried out in a given order, e.g. as in cooking a meal (case B.a). Block serialisable, B.b, is a sub-set of Goals that are not pair- wise independent either with goals in the same sub-set or with any other Goal. However, as a set are serialisable with other Goals that might exist, but themselves form a non-decomposable problem. The general non-decomposable problem is shown in B.c. We have not yet established how best to handle the priorities of a set of non-decomposable Goals. However, for the case of concurrent cooperative Goals the current model will be augmented to increase the priority of all goals in the set if any of the goals is a winner in its respective action-domain. This fact comes from the recognition that it does not make sense to try to approach a table in a suitable pose while the corresponding Grasp Goal in the Arm domain might be suppressed by another domain Goal, for example WaveHand. The Movement and Grasping commands, even though independent in a pair-wise manner, still form a group with dependencies in order to achieve the final Goal of *Grasping an object from the table*. In this case this Goal might be considered as an emergent Goal.

4 Results

The Computational Model in this paper has been tested both in a simulated and in a real robotic agent. Results on the priority of competing Goal families are shown in figure 4. The scenario used is the following: We provide the real robotic agent with a set of spatial goals in a grid [-1,1] x [-1,1] using consecutive commands of the type MoveTo (x,y) (SP1-4). We have used four spatial commands corresponding roughly to the four corners of the grid. After the second MoveTo command a MoveArm (AP1) command was issued asking the robotic arm to be extended in a forward direction as the robot starts executing the movement towards the third goal. While the robot was in transit a series of obstacles were placed in front of the third spatial goal so as to make this inaccessible. All goals of the scenario had equal Commitment, Engagement, EW and W values, so as the Motivation system did not affect them in the observed time scale. We see that goals SP1 and SP2 are executed and terminate sequentially as expected. When SP1 terminates, the Global Attentional competition assumes SP2 as the winner. The interest in this experiment lies in the behaviour

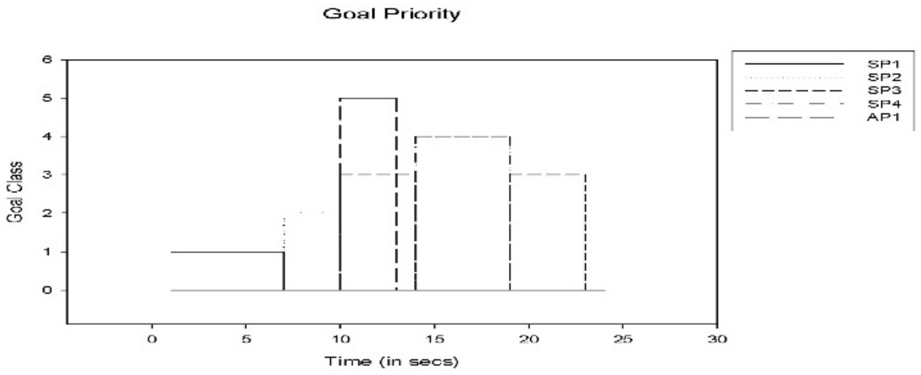

Fig. 4. Goal priorities for spatial goal SP1-4 and arm goal AP1. Duration of experiment is 25 secs for all goals to complete. The duration of each goal reflects the time that takes for the robotic agent to other move into space or extend the arm.

of goal SP3. After the termination of SP2, SP3 assumes execution and in parallel AP1 is executed as well. While the robot is in transit we place some obstacles in front of the SP3 location. The robot remains in the vicinity of SP3 for a while. After some seconds have passed the SP3 termination probability is lowered, thus resulting in SP4 becoming active. SP4 is now normally executed. After termination of SP4, SP3 is resumed again at which point we remove the obstacles and allow the Goal to be achieved and terminate. The interest in this case is the ability of the system to 're-member' given goals and react according to a changing environment. Goals may be suspended and re-activated again without being terminated (in a sensible time-scale). Repeating the experiment a second time with obstacles' removed after a longer time (360 secs) resulted in the constancy of the rate of target convergence and the corre-sponding lowering of the Termination Probability. Eventually the Termination Prob-ability became smaller then the Elimination Probability and the Goal was deleted from the GoalSpace as it should be expected for a goal that does not achieve any progress.

5 Discussion

We have presented a Computational Model that corresponds to the execution compo-nent of our Cognitive Architecture. We made a distinction between a plan coming from the Reasoning System and its actual run-time monitoring which raises re-plan and learning requests among other things. We have discussed in detail the concept of Goal for our architecture and we provided our justification for this approach. We also discussed numerous Goal attributes that influence Goal priority and termination. These attributes provide the necessary integration links with other components of the Cognitive Architecture. Finally we discussed the problem of priorities for classes of plan decompositions. There is still future work in this direction so as to cover all iden-tifiable classes. Finally there is one more issue regarding priority. One can imagine a case where a losing goal can very speedily be satisfied; thus to allow *opportunistic execution*. In this way we will maximise the throughput of the agent. However, one should investigate further if this increased processing throughput will lead actually to a higher motivational level, which is the ultimate cost function that must be optimised. This idea is already implemented in (1) but it needs further work to test its value. One has to make more concrete the expected gain of the approach by studying in concert the computational model together with the motivational system.

Acknowledgements

We would like to acknowledge the support of the European Union through the IST GNOSYS project (FP6-003835) of the Cognitive Systems Initiative to our work.

References

1. Rusell, S., Norving, P. *Artificial Intelligence: A Modern Approach,* Prentice Hall, 2nd Ed., (2003).
2. Korf, R. 'Planning as Search: A Quantitative approach', *Artificial Intelligence* 33, 65-88, (1987).
3. Knoblock, C. 'Learning Abstraction Hierarchies for Problem Solving', In *Proc. of the Eighth National Conf. on Artificial Intelligence*, Boston, MA, (1990).
4. Kasderidis S. & Taylor J. G., 'Attentional Agents and Robot Control', *International Journal of Knowledge-based and Intelligent Systems* 8, 69-89, (2004).
5. Kasderidis, S., Taylor, J.G. 'Combining Attention & Value Maps', *In Proc. of 15th Int. Conf. on Artificial Neural Networks (ICANN 2005),* Warsaw, Poland, Vol. I, Ed. W. Duch et al, Lect. Notes in Comp. Sci, Vol 3696, pp. 79-84.