Proceedings of the 4th International Symposium on INTELLIGENT ROBOTIC SYSTEMS - SIRS '96
Lisbon , Portugal , July 22 - 26 , 1996

Intelligent Control for Perceptually Attentive Agents: The S* Proposal

John K. Tsotsos

Dept. of Computer Science
University of Toronto
Toronto, Ontario, Canada

Abstract

The goal of this research was to reconcile the successful behavior-based robot control architectures with the results presented in (Tsotsos 1995). Those results were critical of the behavior-based methods with respect to their scaling properties and the claim that the mechanism may also explain human behavior. Analysis shows that mechanisms which were anathema in the early behaviorist dogma, such as attention, hierarchical processing, and goal-directed processing, are needed. The S* architecture marries these two points of view. The result is a conceptual structure which may construct more flexible and sophisticated control strategies.

In addition to presenting additional conceptual power in the form of the S* architecture, a unique method for determining whether a particular S* behavior collection can satisfy a given mission is described. A language for mission plan specification permits the integration of deliberative and reactive mission specifications such that all parameters specifying both types of robot behaviors may be dynamically under the control of the mission. A proof procedure is sketched which can determine if a particular behavior set violates the timing and feasibility constraints of a specific mission plan. To our knowledge, this is the first such proof procedure that ties mission plans to particular control architecture instantiations and mission timings.

All of the specifications of S* are preliminary; that is, there have been no experimental verifications of any part described in this document. Thus, stronger statements about the capacity of S* must be deferred until future work completes an implementation.

Previous Robot Control Methods

The sensing, thinking and acting parts of a robot's function must be organized within some framework and must accommodate requirements on the robot's performance such as multiple sensors, uncertain processes, conflicting goals, and planning. Two basic control paradigms have appeared for organizing the control systems of an intelligent mobile robot: functional decomposition (or centralized) and behavior-based decomposition (or distributed). Usually the former is stated in terms of a sense-model-plan-act framework while the latter is given in terms of parallel sense-act layers. Hybrids of these have also appeared.

A static, fixed SMPA model cannot react to events which occur in the robot's world while the robot is 'thinking', that is, modeling and planning. This is the key reason why the strict version of SMPA is inappropriate as a control method for a robot in a dynamic world. The strict version of behavior-based control is also inappropriate. Strict versions do not permit for mission-specific behavior compositions.

There are many robot control strategies; a thorough review of the behavior-based, reactive methods can be found in (Arkin & Grupen 1993) while Thorpe (1992) offers a different perspective on mobile robot control. Also, the review by Iyengar and Elfes (1991) and the proceedings of a workshop on the topic (Chatila and Harmon 1992) provide good sources of background material. Although behaviorist methods, such as subsumption (Brooks 1986) have appeared successful and have led to mobile robots with interesting performance, a strong theoretical argument may be made that a control strategy based on the strict version of subsumption is inappropriate for anything but a very limited domain of behavior. The analysis and proof of this inadequacy can be found in (Tsotsos, 1995). It was shown that if scaling to human-like problems and performance is desired behaviorist schemes must be supplemented with: i) intermediate representations, ii) explicit representations of goals, iii) hierarchical organization and iv) attentive processes. In addition a broader review of the control literature adds the following: v) selective attention is important for reducing computation time and can be applied to a variety of search problems inherent in robot control decision-making (Grandjean and Matthies 1993); vi) the general sense-model-plan-act paradigm appears in most if not all proposals in some form (Fennema et al. 1990); vii) behavior decomposition gives a finer grain specification of the robot's functionality than the monolithic strict SMPA paradigm (Brooks 1986); viii) available computation time and required response time for the robot are important variables and should be incorporated into decision-making (Zilberstein and Russell 1993); ix) grouping of behaviors into clusters is valuable for controlling the space of decisions required of the conflict resolution mechanism (Watanabe, Onoguchi,

Kweon and Kuno 1992); x) a conflict resolution method is needed: priority rules are widely used successfully (Watanabe, Onoguchi, Kweon and Kuno 1992). As a result, these elements numbered i) to x) form the core of S*.

The S* Proposal

Our proposal is one which subsumes the subsumption ideas of Brooks. Here, a behavior is taken to mean any process which uses input available in one or more representations and causes an effect to one or more (may be the same) representations. The representations are defined as part of the agent. S* uses a generalized definition of the 'external world': the world may refer to any representation whether it be one of the internal (logical) representations or the external (physical) world. The world is added to the SMPA cycle to create a sense-model-plan-act-world cycle. Each behavior is represented as an SMPA-W cycle. Behaviors may thus act on the external world, by manipulating physical objects or causing the robot to move, or may act on the internal world of the robot. That is, a behavior may manipulate internal representations, taking input from an internal representation and making changes to or creating another internal representation for use by subsequent processes. Intermediate representations, hierarchical organizations, attentive selection and explicit goals are thus all facilitated.

The SMPA-W Framework

Each behavior, whether internal or external, is represented as an SMPA-W cycle. Although the concept of SMPA control has been criticized in the past, the criticism is due to its use as a single integrated control paradigm. Here, it is used as a means to decompose the stages of computation within individual behaviors. The SMPA-W used is a five-node cycle shown in Fig. 1; the fifth node is the world which provides the input-output definition for the behavior. These behaviors include those which lead to external behavior of an agent plus those which lead to internal inferencing, reasoning or planning behaviors of the agent. The elements are defined as:

World: contains two representations, an event window and an action window (see Fig. 1) . The event window opens up (or makes accessible) a relevant portion of some set of representations within the system. Similarly, the action window opens up onto some set of representations (may be overlapping with the event window). The world node also contains a set of demons which monitor the contents of the event window. These demons detect changes to the event representations of relevant types which act as triggers for the activation of the behavior. The behavior is quiet until the demons awaken it. In this way, the representations are not limited to be those of the external world. Intermediate representations permit internal behaviors, yet are handled in the same manner as external behaviors.

Sense: a process which invokes a particular sampling of an event window based on active triggers in the world node, and creates a representation from that sampling. On activation of the behavior, the sense node extracts from the event window the trigger stimuli and may further process the information.

Model: using a representation of a domain and the sensed representation, a model is associated with subsets of the sensed representation.

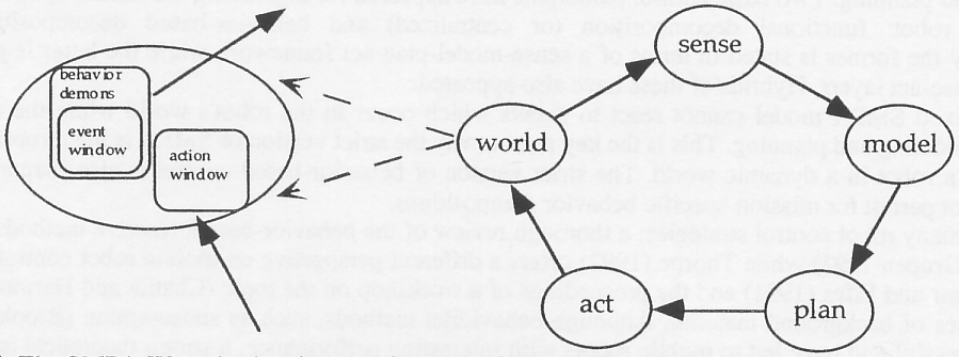

Figure 1. The SMPA-W cycle showing the elements of the world node.

Plan: a process which associates behaviors with models and determines the appropriateness of the association.

Act: a process which invokes some manipulation of a representation that realizes a behavior. All effects of the act node are seen within the action window of the world node.

This is not a strict cycle; only if the event and action windows open onto the same representation would this be so. Nevertheless, a behavior may be a continuous one; it might upon completion wait for the next triggers to appear in the event window. A single-shot behavior would terminate on completion of its act function.

This concept may be applied recursively. Each node in the cycle may be augmented by one or more other cycles. For example, suppose that there are different versions of optic flow computations, one which is quick but not very accurate another that is more accurate but slower. The latter would augment the former computation; a conflict resolution stage would determine which one to use when depending on task demands. Requirements for speed may

lead to the former choice while a requirement for accuracy would lead to the latter. The structure within a node is shown in Figure 2.

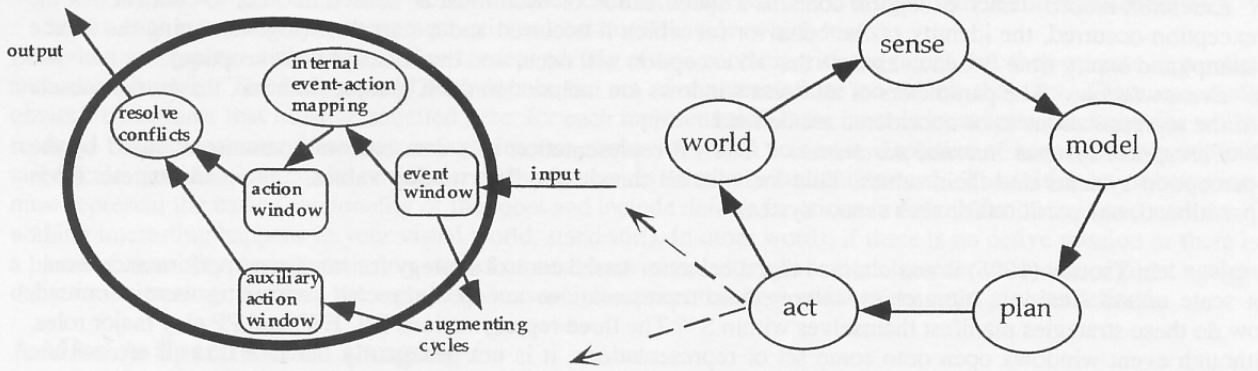

Figure 2. The internal components of each node within the SMPA-W cycle (other than the World node which was defined in the previous figure).

Each of the S, M, P and A nodes of the cycle contain the following components, shown in Figure 2 :

event window: pointers to subsets of relevant representations which contain the information necessary for the node's function.

action window: pointers to the subsets of representations that are affected by this node's computations.

auxiliary action window: a duplicate copy of the action window which is affected by the action of an augmenting cycle which uses the same event window. There may be more than one of these; one is needed for each augmenting behavior. This is so because there is a need to compare the results of each augmenting behavior with the results of the augmented behavior in order to choose which results to pass on.

internal event-action mapping the processes by which the node performs the computation for which it is responsible in its SMPA-W cycle.

resolve conflicts: if there are one or more augmenting behaviors contributing to this node, then the output of the node to the cycle must be resolved. This component implements a local priority network for performing this decision. The priorities may be altered dynamically depending on current task demands or state of interpretation.

Event windows may dynamically change; action windows are static.

The definition of an augmenting behavior follows: an augmenting behavior is a behavior which uses the same event and actions windows as one of the S, M, P or A nodes of some other behavior. The processing done by the augmenting behavior may provide more precise or alternate descriptions.

Note that although all nodes in the control architecture have both event and action windows, it is not the case that information is duplicated all over the network. Rather, the windows may be considered as pointers into a representation stored elsewhere. Since several windows may open onto the same representations, there must be a way of ensuring consistency within each representation. Standard methods may be used here to accomplish this.

Representations for a Typical S* Architecture

A number of representations would be required in any typical robot control application. A minimal set may be the following:

R0 state This representation contains the current state of the robot (position, orientation, camera and head information, power, global clock, etc.). It is assumed that the robot contains registers or buffers into which commands are written for execution and are the direct access to the robot's actuators. Further, it is assumed that there are registers which an external device may read in order to extract the status of the robot's functions (encoders, power level, etc.).

R1 actuator commands This representation contains the requests for robot motion that arise from all behaviors. Each must be time-stamped (start and expiry time) and must have an owner (which behavior made the request).

R2 mission/task direction This contains the representations of the mission characteristics that come from the operator. This also contains any sub-goals determined by the behaviors to be necessary to achieve in order to satisfy the mission. Information relevant to the detection of mission completion must also be included.

R3 environment R3 contains the representations of the environment that the robot is in. This may include a map, models of landmarks, models of other known objects, etc.

R4 recognized aspects of environment This representation stores features, edges, regions, object surfaces, optic flow, model fits, recognized objects, etc., detected by the perception system (all sensors). It is not only for current image results but also includes the integration of results over sequences of images.

R5 sensed data These are the raw signals from the robot's sensors, context stamped. Since this has the potential of being a very large data set, some mechanism for discarding signals must be included.

ER Exception record Each exception contains a specification of what must be sensed in order to confirm that the exception occurred, the identity of the behavior for which it occurred and a start time (together being the source stamp) and expiry time (when to assume that no exception will occur and thus discard this exception).

EW Event windows The parameters of all event windows are included in the EW representation, that is, the subsets of the representations to be considered are defined.

PP Perception systems internal parameters The PP representation is a data base of parameters used by the perception systems and their values. This includes all thresholds, filter tuning values, any constants, etc. PP is partitioned, one partition for each sensor system.

In Tsotsos (1995) it was claimed that a behavior-based control strategy for intelligent performance would not scale unless attention, hierarchies, intermediate representations and goal-directed processing were permitted. How do these strategies manifest themselves within S*? The three representations ER, EW, and PP play major roles. Although event windows open onto some set of representations, it is not necessarily the case that all of each the representations need be considered at each behavior invocation. This is especially important since behaviors are activated by a set of demons which scan the event window for triggering events. Thus, any reduction of the search space is valuable. The parameters of all event windows (subsets of representations) are included in the EW representation. Event windows are dynamic and open up onto selected portions of representations. The parameters which control event windows are themselves stored in the EW representation onto which an action window may be opened. Thus, an attention behavior (or several of them) may dynamically modify the event windows of other behaviors depending on the state of recognition or navigation progress.

A typical perception system contains many parameters, ranging from required thresholds, tuning constants of filters, to event window parameters. In an attentive selection scheme, many or all of these may be altered dynamically during the course of recognition in order to optimize the process. The PP representation is a data base of such parameters and their values. When a particular perception behavior is triggered, the current values of the parameters it requires are read in before the SMPA-W cycle is executed. The PP representation may be part of an action window for some behavior and thus the perception process itself may be tuned.

Intermediate representations appear within the event and action windows as representations of actions which affect the internal world of the agent. Explicit representations of goals are included in a 'mission' representation and may be used by any behavior (as part of their event window) and may be used to target processing by any behavior. Hierarchical organization of any kind may be constructed using the flexibility inherent in the event and action window structure.

Goal-directed processing also requires a method for the detection and correction of deviations from the goal. Each 'plan' node has an additional special function. Each adds a list of potential exceptions to the exception record representation. That is, these are potential failures which might occur during the execution of this behavior. Each exception contains a specification of what must be sensed in order to confirm that the exception occurred, the identity of the behavior for which it occurred and a start and expiry date (when to assume that no exception will occur and thus discard this exception). Each exception must have a behavior which can deal with it.

A Rough Guide to Building an S* Control Structure

It must be clear that the intent of S* is not to provide an optimal controller in the control theoretic sense. Rather, S* is intended as a conceptual framework which seeks to remedy the representational shortcomings of the subsumption architecture. This section presents a guide to building an S* structure for a particular robot. The guide suggests which behaviors and representations must be defined as a minimal set.

A Generic S* Structure for a Robot

The starting point is the definition of all the representations, behaviors, exceptions and trigger events for each behavior. As stated earlier, the representations for exceptions (ER), perception parameters (PP) and event windows (EW) are mandatory in S*. A high level procedure for defining the behavior set is:

A) Enumerate actuators of robot.

→For each, define a behavior to arbitrate the requests to it and define a representation for those requests.

Enumerate the classes of actuator actions

→For each actuator action, there must be at least one behavior which places a request for that action

B) Enumerate the sensors

→At least three representations are needed for each sensor: one to hold the raw sensed data, one to hold the interpretation of that data , one for the perception parameters (a portion of PP).

→For each sensor, at least one behavior must be present to provide an interpretation of the data

→For each sensor, there must be at least one behavior which sets the perception parameters for that sensor.

C) For each behavior, enumerate all exceptions

→Each exception requires at least one behavior which can deal with it.

D) For each behavior, determine the trigger events and their demons

→For each trigger there must be at least one behavior which writes the trigger into some representation

E) For each representation, there must exist at least one behavior which reads it

Behaviors and representations are defined independently of one another, with the intent that they operate in parallel and autonomously. The link among behaviors is the set of representations on which they act. There are some obvious constraints that must be satisfied here: for each representation there must exist at least one behavior which reads it; for each actuator there must exist at least one behavior which acts on it. Insertion of new behaviors without large-scale modifications to the remainder of the system is relatively straightforward. The initial set of behaviors must represent the basic functionality of the robot and include default functions (a default function is one such as: if nothing interesting happens in your visual world, stand still). In other words, if there is no active mission or there is a pause of inactivity during a mission, the robot must have some active behaviors in order to remain vigilant against dangerous situations. Further, the representations which support this core of functions must also be defined.

A Mission Specification Language

Execution of a mission requires a first stage to define the task steps. The intelligent agent task is more difficult than single-mission systems since the agent must be capable of executing a wide variety of mission types. If the mission specifies that the robot must move from one point to another on a map, then a simple plan is possible in terms of the motions that the robot must execute (say in terms of move forward, left, turn, etc.). Also, the planner would know the kinds of position uncertainties inherent in the robot's movement, and could explicitly include position verification and correction steps into the plan. Reactivity is not so easily captured by the STRIPS-like list of plan steps implied here. What is really required is a new method for specifying a plan that includes not only the motions of the robots, but also specifies the aspects of the environment of which the robot must be vigilant while executing the plan. It includes the ability to specify behavior sequences, parallel behaviors, timings among behaviors, timing constraints on behaviors or set of behaviors, and parameters to behaviors. For example, a hypthetical mission plan might be:

```
start < 75 min
phase P1 {   find landmark ( D );
             move forward ( 10 feet ) within track landmark ( D );
         turn left }
phase P2 {   move forward ( 25 feet );
             find landmarks ( A, B and C ) starts with verify position ;
         move forward ( 30 feet ) ; }
phase P3 {   turn right ;
         move forward ( 5 feet ) }
while W1 { P1
  while W2 { P2
       ensure (2/min, ¬yield-right-of-way && position-uncertainty <1%
                  && avoid(fuel spills)) }
  P3 < 30 min
  ensure (5/min, yield-right-of-way && position-uncertainty < 5% ) }
end
```

using the grammar

```
mission → start time behaviors end
behavior-atom → behavior-id (p-list) | phase-id | while
behaviors → behavior-atom | behavior-list | behaviors behavior-connector behaviors
phase → phase phase-id time { behaviors }
while → while while-id { behaviors ensure (rate, Boolean) }
rate → null | rate-specification
time → null | time-specification
behavior-connector → ; | temp-connector
behavior-list → [ b-list ) | ( b-list ]
b-list → behavior-atom | b-list , behavior -atom
temp-connector → starts with | within | during | starts after time | ends with | ends during
behavior-id → { system behaviors }
p-list → parameter | p-list, parameter
parameter → { possible system parameters }
```

For further detail on the misson language see (Tsotsos 1996).

From Mission Specifications to Behavior Sets

How can a set of behaviors be selected to satisfy a particular mission specification? Suppose a mission must be completed in time τ. This might be a more difficult problem if it were not for a particular simplifying tactic: the mission specification language uses the behaviors themselves as part of the language; this yields the initial set of members to the behavior set B. Although this solves part of the problem, it does not solve all of it. The behaviors specified in the mission might require many additional supporting behaviors in order to properly execute. A simple method may suffice for determining which those supporting behaviors are:

Repeat until no new behaviors are added to set B

For each behavior, enumerate 'world' node triggers (including representations where they are to be found)

For each, locate the behaviors which provide those triggers to those representations

Behaviors that provide trigger events must be ordered in time; this ordering would be at best a partial order. The ordering must have links into the mission sequence so it is known when a behavior's trigger events might be available. Thus, a network can be created whose nodes are behaviors, each node is labeled with its worst-case cost, and directed links between nodes show the sequence of trigger events between behaviors.

It is important to show that a given behavior network, derived from a mission specification, can actually satisfy the timing constraints imposed by the mission. The proof-theoretic procedure below accomplishes this on the assumption that no exceptions are raised during the execution of any behavior set. Clearly, if exceptions arise, no timings can be predicted.

Each behavior b has associated with it the following:

$C_b(e_b)$	cost (in time) of computation as a function of size of input e_b (event window)
W_b	cost (in time) of writing to action window
A_b	ncremental additional cost (in time) of augmenting behaviors (plus conflict resolution)
$e_{b_{min}}$	minimum event window size required (defined by recognition targets, imaging parameters).

The costs are all worst-case (or upper bounds) estimates which must be derived by analysis of the algorithm and code included in each behavior. This does not take into account any time used waiting for a trigger event or for other data to be provided by some supporting behavior nor should it; the network structure includes this implicitly. The amount of time which may be wasted waiting for write permissions is not included. It will be assumed zero.

The reason why this particular formulation was chosen has its roots in the reason for wanting to enhance subsumption in the first place. The arguments laid out in (Tsotsos 1995) were based on computational complexity and demonstrated that the subsumption architecture was inadequate because it ignored complexity issues particularly as they pertain to sensory processing. It seems natural then to include complexity directly in S* in this manner in order to overcome this inadequacy.

The specification of $e_{b_{min}}$ is motivated by the study of 3D search behavior in Ye & Tsotsos (1994). There, a detailed algorithm is presented which outlines how visual search might take place in an unknown and large 3D space. One of the important sets of parameters required by the algorithm is the field of view in relation to the target sought. Any recognition algorithm must be presented with a reasonable projection of potential targets in images in order to have some chance at detecting them.

The total time to run a given behavior is $t_b = C_b(e_b) + W_b + A_b$. Most behaviors seem to have a function C_b which is either constant or has very small variation with input size. This is particularly true in situations where the input is some small list of information. The function achieves its full importance if sensor data is concerned and where search over large spaces is required. e_b is the actual event window size required and is dependent on the actual targets specified for the behavior.

If a specified task T must be accomplished within a particular time interval τ, it is possible to determine for a given set of behaviors what event window sizes might be required. It is even possible that a particular behavior set can be rejected as being infeasible for the task. Further, a hierarchy of timing constraints may need to be satisfied determined by the mission specification. Thus, not only must the overall constraint τ be satisfied, the constraints on each subset of behaviors also must be satisfied. Let the set Δ be the set of behavior subsets, the i-th element is denoted by δ_i each with their own timing constraint τ_{δ_i}. Then, the following must be satisfied:

$$\forall i,\ \delta_i \in \Delta,\ \tau_{\delta_i} \leq \sum_{b \in \delta_i} t_b$$

Strictly speaking, this assumes that the behaviors in each subset are single-shot behaviors in sequence. This is not in general true; the behaviors may form a network and in this case, the longest path is what is actually needed above. Even so, the above does provide a conservative estimate on this constraint.

Suppose that a set of behaviors B has been selected to satisfy T, that a network is created whose nodes are these behaviors, and each node is labeled with cost using $e_{b_{min}}$. Let the longest path within this network of behaviors be Φ. Although the general problem of finding the longest path in a graph is NP-Complete, polynomial

time algorithms exist for acyclic directed graphs (Lawler 1976); Φ is such a graph. It is possible to compute an overall minimum (but not actual) duration D for the execution of the task in terms of input size for each behavior:

$$D = O(B) + \sum_{b \in \Phi} C_b(e_{b_{min}}) + W_b + A_b$$

O(B) is the overhead required for execution of the overall system as a function of the behavior set (event demons, database accesses, real-time operating system, etc.). Although an estimate for the duration of the set Φ may be found by using the times to execute determined by $e_{b_{min}}$ for each behavior, when a particular e_b is assigned (as a result of target or trigger event information), the duration will change.

A third complexity constraint can also be stated based on the network linking the behaviors of set B. Any real system will be constrained not only by time but also by space. In particular, the number of behaviors which must be active in the system at any time must be such that the available processors and memory are not exceeded. For this discussion, it is assumed that each behavior requires a dedicated processor and that the processor must have a specified amount of memory for the execution of the behavior. Assume that the set of processors available is P. At any point in the behavior network, it is possible to identify which behaviors are expected to be active; each active behavior has an expected execution time t_b which will be a function of the behavior's $e_{b_{min}}$. Further, the memory requirement of each behavior will also depend on $e_{b_{min}}$ for that behavior. Partition the network of behaviors into sets such that each set contains either a single behavior that executes or a set of behaviors which execute in parallel. Let the set of these sets be Σ. The number of processors needed would be simply the cardinality of the behavior set s, where s is an element of Σ. Note that $\bigcup_{s \in \Sigma} s = B$. Thus, the maximum number of processors required for the mission may be given as $P = \max_{s \in \Sigma} (|s|)$ while the memory requirements are: $M = \max_{b \in B} (R(b, e_b))$ where R is the amount of memory behavior b requires when its event window is of size e_b

The next step is to determine the best values of e_b using the specific targets and trigger events given in the mission. If, in order to satisfy the time constraint, the size of any behavior's event window must be smaller than the corresponding $e_{b_{min}}$ for that behavior, then the set B is not a feasible solution for T. The search problem is then re-cast as the following:

1. choose the elements of the set B and determine the sets Φ (based on $e_{b_{min}}$), Σ and Δ.
2. ∀b ∈ B, find e_b
   a) such that $e_b \geq e_{b_{min}}$
   b) ∀i, $\delta_i \in \Delta$, $\tau_{\delta_i} \leq \sum_{b \in \delta_i} t_b$, (defined using e_b)
   c) for D defined using e_b, $D \leq \tau$ and is as small as possible
   d) $P \leq P_{max}$
   e) $M \leq M_{max}$

There may be many solutions since all large values of e_b may be acceptable. The constraint on D being as small as possible will eliminate these trivial solutions.

This is the verification procedure for a particular S* solution to a particular mission (specification). A solution may be obtained for this problem via dynamic programming; the full optimization problem is likely NP-hard. Note that this procedure can only be used to conclude that a particular behavior set B *cannot* satisfy the timing constraints. If the set B passes this test, the only conclusion that can be drawn is that the resulting system *may* satisfy the timing constraints. The fact that exceptions cannot included in this procedure is the main reason for this. It is important to note that this does not check the correctness of the behavior set for the specific mission.

Comparisons to other Formal Methods

This contribution of this work is the use of complexity-theoretic time measures in the specification of a behavior and in the verification of behavior sets for a particular mission. Previous schemes use a variety of mechanisms for dealing with time. Some employ the 'synchrony hypothesis': this assumes that the system instantly reacts to external events and does not take into account the finite time required for computation. 'Statemate' is an example of a popular tool which makes this assumption. Another tool is the use of upper and lower bounds on time specifications, such as are used in Timed Petri Nets. Here, events may occur anywhere within an interval and still be correct. Logical formulations (using formal logic) also typically employ time bounds, and many also use external clocks to which the bounds are tied.

Formal methods usually must assume that the external world is known, closed and totally predictable. This is not the case in real situations. Otherwise, the proof procedures cannot be applied. It is interesting to note that in all cases, these formal methods lead to verification procedures which are provably exponential (oven doubly exponential - see Ostroff 1992 for a useful comparison). In order to make matters worse, none of the formal methods

makes the proper assumptions regarding perception. That is, perception is assumed to be at worst a linear problem, when in fact, it might be quite a bit more expensive if not also exponential in nature.

In the above method for S* a new timing tool is proposed which acknowledges that behaviors have different time requirements for different tasks. In other words, rather than using fixed lower and upper bounds irrespective of the task, S* permits a more fine-grained analysis of timing requirements by using a complexity-theoretic measure which is specific for the amount of input required for a particular task.

Although a complexity-theoretic timing bound has important advantages, it does not change the fundamental exponential nature of the proof process. Here, the optimization problem is cast into a format where dynamic programming may find approximately optimal solutions. The procedure simply checks to see if a particular behavior network derived from the mission specification satisfies the timing constraints. Search for the optimal behavior set is outside this procedure.

Conclusions

A new conceptual structure for intelligent, perceptually-attentive, robotic control has been presented. The strategy was motivated by the current successful methods (such as subsumption) but also by a theoretical analysis of those methods which showed that many of the computational mechanisms previously thought unnecessary are critical. These include visual attention, intermediate representations, hierarchical processes, and goal-driven processing. The S* architecture facilitates these mechanisms within a behavior-based framework. The behaviors of S* are generalized in the sense that they may operate either on the physical world or on internal representations and these types of behaviors are treated uniformly.

In addition to presenting additional conceptual power in the form of the S* architecture, a unique method for determining whether a particular S* behavior collection can satisfy a given mission is described. A proof procedure is sketched which will confirm or deny that a particular behavior set satisfies the timing and feasibility constraints of a specific mission plan. To our knowledge, this is the first such proof procedure designed that ties mission plans to particular control architecture instantiations and guarantees mission timings.

All of the specifications of S* are preliminary; that is, there have been no experimental verifications of any part described in this document.

Acknowledgments

This research was funded by PRECARN Associates Ltd., Technology Ontario, Ontario Hydro and AECL Ltd.

References

Arkin, R., Grupen, R., (1993). Behavior-based reactive robotic systems, Tutorial Notes, IEEE Conference on Robotics & Automation, Atlanta.

Brooks, R., (1986). A Layered Intelligent Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation RA-2,* April, p 14-23.

Brooks, R., (1991). Intelligence without representation, *Artificial Intelligence 47,* 139 - 159.

Chatila, R, Harmon, S (eds.) (1992). IEEE Workshop on architectures for intelligent control systems, May, Nice.

Fennema, C., Hanson, A., Riseman, E., Beveridge, J., Kumar, R., (1990). Model-directed mobile robot navigation, *IEEE Trans. Systems, Man, & Cybernetics 20(6)*, p 1352 - 1368.

Grandjean, P., Matthies, L., (1993). Perception control for obstacle detection by a cross-country rover, Proc. IEEE Int. Conf. on Robotics and Automation, Vol. 2, p 20 - 27.

Iyengar, S., Elfes, A., (1991). Autonomous mobile robots: Control, Planning and Architecture, IEEE Tutorial Press.

Lawler, E.L., (1976). **Combinatorial Optimization: Networks and Matroids**, Holt, Rinehart,Winston, New York.

Ostroff, J., (1992). Formal methods for the specification and design of real-time safety critical systems, *J. of Systems and Software 18-1*, p33 - 60.

Thorpe, C., (1992). Robots, Mobile, in **Encyclopedia of Artificial Intelligenc**e, 2nd Ed., edited by S. Shapiro, John Wiley & Sons, p. 1409 - 1416.

Tsotsos, J.K. (1995). On Behaviorist Intelligence and the Scaling Problem, *Artificial Intelligence 75,* p 135 - 160.

Tsotsos J.K. (1996) . S*: Intelligent Control for Perceptually Attentive Agents, RBCV-TR-96- June 1996, Dept. of Computer Science, University of Toronto.

Watanabe, M., Onoguchi, K., Kweon, I., Kuno, Y., (1992). Architecture of behavior-based mobile robot in dynamic environment, Proc. IEEE Robotics & Automation, Nice, France, p2711 - 2718.

Ye, Y., Tsotsos, J.K., Sensor planning in 3D object search, RBCV-TR-94-47, Dec. 1994, Dept. of Computer Science, University of Toronto.

Zilberstein, S., Russell, S., (1993). Anytime sensing, planning and action: A practical model for robot control, Proc. IJCAI, Chambery France, p 1402 - 1407.