Robot middleware must support task-directed perception

Andrei M. Rotenstein, Albert L. Rothenstein, Matt Robinson, and John K. Tsotsos

Abstract—**While current robot middleware projects address functional concerns of robot control, middleware should also support the needs of task-directed perceptual processing. S* is an approach to intelligent control that addresses this concern in particular. It prescribes constructing intelligent controllers as behaviour-based systems augmented with shared representations, which supports task-directed perception and also enables the development of extensible controllers. The focus of this paper is on a software framework, developed to support the development of intelligent robot control systems by teams of designers.**

I. Introduction

With regards to developing middleware for robot control software systems, it is not enough to focus on decoupling functional modules. Designers must also provide support for the particular requirements of the *task-directed perceptual processing* within a robot control system.

Current work has focused on developing middleware that enables the integration of different kinds of modules responsible for *functional concerns*. A functional concern in robotics is a problem of perception, decision or action in robot control that has an algorithmic solution. Examples are collision avoidance, navigation, landmark detection, object recognition, localization, mapping, manipulator kinematic models, locomotion, and speech synthesis, etc. The current focus has been on organizing functional modules as loosely-coupled communicating processes, which execute in a distributed fashion on a cluster of computers. Excellent examples of such middleware are MIRO [30], MARIE [9], OROCOS [8], YARP [16], CLARAty [19] as well as many others. Another notable entrant is Microsoft [18]. While it is necessary to decouple such modules, however, it is not enough to focus on functional concerns: researchers must also deal with extensibility issues imposed by the robot control architectures they develop on top of their middleware. One of these constraints relates to how *perceptual processing* is organized in the architecture.

The hybrid approach [14] to building control architectures, which is undoubtedly the most popular approach today, imposes constraints that make it difficult for a designer to introduce new kinds of task-directed perceptual processing. *Task-directed perception* is any perceptual processing, such as visual search or navigation, which depends on information relating to the robot's current task at-hand; this information may change as the task changes, but the perception system itself remains the same. This is not to say that it is not *impossible* to support new kinds of such processing; rather, a designer potentially needs to ignore his architectural abstractions in order to do it. Thus, the system design is violated. From the perspective of software engineering, this control solution is not extensible.

Manuscript received December 18, 2006. The authors are with the Centre for Vision Research and the Department of Computer Science and Engineering, at York University, 4700 Keele St, Toronto, Canada, M3J 1P3. John Tsotsos is a Professor and holds the Canada Research Chair in Computational Vision. Andrei Rotenstein and Albert Rothenstein are graduate students. (Andrei Rotenstein is the corresponding author; he can be reached at `andrei@cse.yorku.ca`).

In this position paper, we present an approach to robot control that is extensible with regards to supporting task-directed perception. Middleware that supports such extensions must be concerned with aspects of intelligent *architecture* (in the sense of cognitive systems) as well as functionality. From the perspective of software engineering, robot control problems have the special distinction of requiring tight integration into the operating environment, which, in turn, requires perception of the environment. To be efficient, a robot's perceptual processing must be tightly woven with other parts of the control system. Yet, artificial perception is a problem that is far from solved today, so control system designers need new software design principles that allow new and better perceptual processing to be incorporated into existing control systems. Indeed, developers of conventional, non-robotic software systems typically do not have to deal with a problem like this.

We first discuss in the Background Section (II) how hybrid controllers limit the incorporation of task-directed perception. Our solution to this problem involves applying a novel, behaviour-based approach to intelligent control, called S*, that supports the concern of task-directed perception, and this is presented in Section (III). We present an example of a successful S* controller in Section (IV), and discuss our software framework for S* in Section (V). We conclude with mention, in the last Section, of our work in-progress to develop a multi-platform distributed software framework.

II. Background

A. Hybrid deliberative/reactive controllers.

Ideally, a hybrid controller, as was best articulated by Arkin [2], is an organization of two types of modules: a deliberative one and a reactive one. The deliberative one consists of a hierarchical organization of 'good, old-fashioned' reasoning algorithms and other planners that ultimately maintain a world model. The reactive one is a behavior-based controller, typically, in the strict sense of Brooks [5], [6], [7], Arkin [1] and others. Recall that this strict understanding of behaviour-based controllers holds that architectures consist of a set of concurrent, independent processes that accept perceptual information as input from external perception systems and rapidly compute candidate actuator responses; these candidate responses are reconciled via an arbitration module to produce a net output that affects the actuators.

In this idealization, behaviour-based processing and deliberation are considered to be fundamentally incompatible but also necessary for intelligence. It is claimed that this apparent paradox is resolvable by integrating these two modules via an *interface strategy*. Arkin described various strategies, such as the *selection strategy*, in which a sequencer turns each step of a linear plan into parameters for the reactive module, or the *advising strategy*, in which the sequencer reactively selects which action is best suited for a given plan step, given the current state of the environment. They ultimately amount to task execution in the sense described by Firby [11] and Gat [12], whether the plans are linear, conditional, a list of intentions generated by a Belief Desire Intention model [13], or some other plan representation. This mediation is handled by the *executive* [12], [29], a module that sits as a middle layer between the two modules. The various layers of computation thus vary from being more abstract or symbol-oriented (thought to be the nature of "deliberative"), the higher it is, to more reactive or signal-processing-oriented, the lower it is. Perceptual and motor information move "up" and "down" through the layers, respectively, from/to the reactive module, with which sensors and actuators are connected. Of course, in practice, reactive modules are never strictly behaviourist, and there is a certain degree of reactivity in the deliberative layer.

B. Hybrid controllers and the necessity of directing perceptual processing using task information

For a control system to be called intelligent, it goes without saying that, generally, it must perform some sort of *perception*, *decision*, and *action* in its environment. With regards to perception, generally, they must employ perceptual processing that supports a wide variety of agent tasks. What may not be obvious is that this requirement implies that, in carrying out some task, the *decision* component must supply information relevant to that task to the *perception* component. The task information can be used as part of processes to filter out sensor data irrelevant to that task and/or direct it to sensor data that is relevant; such processing is *attention*.

As Tsotsos pointed out [26], task information is required at least for attention in perceptual processing: without attention mechanisms that are directed by information relevant to the perceptual task at-hand, at least one well-understood perceptual problem, *undirected* passive visual search, is *NP-complete* and computationally intractable in practice. Suppose that the control system was composed of a set of visual search behaviours, each with task information hard-coded in *tacit* representations. Then, a *biological* behaviour-based system having "human-level intelligence", as claimed by Brooks [5], [6], would necessarily require more permutations of visual-search behaviour modules than there are neurons in the human Central Nervous System [27]. Given that biological visual systems are clearly capable of visual search, it must be that they necessarily employ task-directed attention. The human visual system is therefore an existence proof of the general necessity for maintaining representations of some sort of task-related information in intelligent control systems.

From an architectural perspective, architectural structures must exist that permit the *decision* component to provide task information to the *perception* component. However, both in the idealized model of hybrid architectures and in most practical implementations, controllers do not influence perceptual processing at all. In hybrids, the concern of perceptual processing is addressed as a module organized externally to the rest of the controller. In most examples of control systems reviewed by Arkin [2], modules that accept sensor input (such as from sonars, laser rangefinders, stereoscopic cameras, etc.) and process it (to estimate pose, to localize targets visually, to detect physical boundaries, etc.) are not under any influence of the deliberative module or the executive. In a particularly illustrative example of an executive, the RAP system of Firby [11], allows for the scheduling of sensing actions, but these essentially query a database of fluents that is maintained by a separate perceptual process that itself is not under the influence of decisional processing. A similar situation exists in all execution systems surveyed in [29].

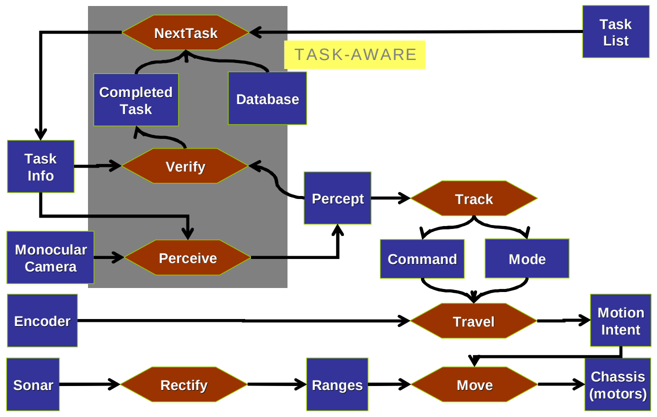

Figure 1: An example of an S* controller, used for visually searching for and physically visiting textured targets in a closed space. The blue boxes are representations and the red hexagons are behaviours. Behaviours have arrows directed *to* their action representations (which store their output) and have arrows directed *from* their event representations (which provide their input). The Perceive and Rectify behaviours perform perceptual processing; The Track, Travel and Move representations perform types of actions. The Verify and NextTask behaviours work together to perform simple linear sequencing. Notice that the Perceive behaviour uses the Task Info representation to perform the processing required for visual search, the result of which is stored in the Percept representation. The NextTask behaviour writes information about the target to search for in Task Info, in order to tune the visual processing to search for a different target.

To summarize, intelligent control architectures should support sharing of task information. Middleware can provide this support.

III. The S* Framework for Behaviour-Based Control

A. The formalization of representation in behaviour-based systems

S* is an approach to building intelligent control systems in the behaviour-based paradigm. Proposed by Tsotsos [28], it is a set of design principles that prescribes support for representation in behaviour-based controllers. Figure 1 depicts an S* controller.

While, in practice, behaviour-based systems are considered necessary components of intelligent systems due to their high reactivity, their lack of representation remains a fundamental theoretical limitation. The strict notion of behaviourism holds that explicit representation should be avoided in intelligent systems. However, this idea has been largely refuted, since representation is absolutely necessary for learning, cognition and perception. As mentioned earlier, task information is necessary for visual search, at least. Yet, strictly behaviour-based systems do not allow the explicit representation suitable for maintaining information relevant to the task at-hand. Claims about behaviour-based systems as scaling to human-level intelligence are therefore invalid [27]. Also, as Kirsh pointed out, representation is required for reasoning, planning, prediction, for adopting reference frames that are non-egocentric (often required for performing various motor tasks), for training pattern classifiers, and so on [15].

While these problematics are widely known today and representation shows up, in an ad-hoc manner, within many reactive systems being built currently (as evidenced by the RoboCup competition systems [3], [4], it is *not* ad-hoc in S*. Support for representation is formalized. S* behaviours do not merely interact with perceptual modules and an arbitrator. Instead, they interact with shared representations, which are (stateful) shared data structures (abstract data types, which include public interfaces consisting of functions and procedures) having single-writer multiple-reader access semantics. Representations might store data such as maps, plans, trajectories, an object database, and so on. Or, representations can be tied to sensors and actuators (they are said to "represent" them), and the act of reading and writing to them is the act of sensing and acting, respectively, in the physical world. They are given names that refer to what information they store (e.g. "map", "sonar", "camera image", "disparity map", "heading", or "attended object").

Each behaviour maintains references to a set of representations, some of which are used as input (said to be its *event representations*) and some of which are used for output (said to be its *action representations*). Behaviours are behaviour-producing modules in the usual sense: they might be called "avoid", "move", "search", "localize", "build map" and have the obvious effect. The behaviours themselves can be designed as *sense-model-plan-act* (SMPA) processes in the traditional sense: each behaviour senses a subset of its event representations; it may construct an internal model suitable for analysis; it may perform some reasoning or other symbolic processing to generate a plan; then, it carries out the plan by writing something to at least one of its action representations. Associated with each behavior is a daemon that maintains a "window of attention" onto a relevant subset of its event representations, and monitors it for relevant state changes. Normally, behaviours are idle. Upon such a detection (termed an *event*), if the behavior is idle, the SMPA process is triggered to execute. Once the SMPA process has finished executing, the behaviour returns to an idle state. The event has a condition associated with it (termed a *trigger condition*), which is a first-order predicate evaluated by the daemon whenever an operation is performed on an event representation.

In this way, each behaviour produces information that may be consumed by others; the organization of behaviours and representations as an example of a *behaviour network.* Figure 1 depicts a behaviour network, which will be further described in Section (IV).

B. *Support for deliberation and attentive processing*

S* gives the designer the ability to implement processing that involves deliberation in a unified, reactive system. Deliberative processing can be designed into each behavior. The usual concern about deliberation is that it impedes a controller's reactivity. However, in our approach, behaviors that consume the output of a highly deliberative behaviour are still likely to be triggered by changes in the action representations of another, more reactive behavior. For example, in a hypothetical scenario, a highly deliberative path-planning behaviour might take thirty seconds to write a new path into a representation that holds it. A "Move" behavior responsible for following it while avoiding obstacles might read a "Sonar" representation, as well. Because the sonar information is frequently updated, the "Move" behavior will be triggered frequently. As long as its execution time is fast, the system's overall reactivity to obstacles will not be limited by the slow planner.

While this may appear to be the same as a hybrid architecture, as described in Section II, it is not, because S* does not prescribe deliberative processing from occurring in a low-level layer or signal processing from occurring in a high-level layer. Nothing precludes the designer from introducing another behaviour such as, say, "Select Goal Location", that produces information to be used as input by the planner behavior and is therefore at a higher level of symbolic abstraction, but that also happens to be highly reactive. Indeed, it is possible for purely symbolic processing to have fast execution time. For this reason, S* does not impede the designer from doing away with the hierarchical organization of a hybrid control architecture if he so chooses.

This kind of interaction is particularly useful for supporting perceptual processing. S* also gives the designer the ability to implement processing that involves attention, especially visual attention. In our current implementations of S*, behaviours that perform task-directed perceptual processing typically have two event representations: one is sensor data, such as a camera image; and the other is task information. For example, in the same hypothetical scenario, suppose the perceptual task is to search the visual scene for a target having a given texture, say, squares of different colors having striped patterns of different orientations. The task information could consist of the color and of the orientation of the striping. To initiate execution of the search, another behaviour updates the task information representation with a new color and/or orientation. The output of the visual search behaviour might be an estimate of the location of the target, which may be required by the hypothetical "Select Goal Location" behaviour mentioned earlier. Indeed, the concern of selecting task information to write to this representation can be viewed as high-level because of its symbolic nature and the concern of performing the visual search can be viewed as low-level because it performs signal-processing. Yet, S* permits the designer to have the visual search process provide information to another high-level process[1].

IV. An example of a successful S* controller.

We now present a motivating example. One application of the S* approach was to develop a behaviour-based controller for a mobile robot (the Argo rover [10]) that navigated an enclosed laboratory, visually searched for textured targets, from a sequential list of targets positioned on the laboratory walls, and physically "visited" them close-up. A controller was implemented using a software framework, to be described in Section (V). It consists of the following set of behaviours and representations, which are organized into a behaviour network as depicted by Figure 1.

Behaviours

Move. A fuzzy-logic controller that computes collision-free platform motion, given rectified sonar information and the system's intended (desired) motion to perform. Effectively responsible for motor speeds.

Perceive. Given information about the target's color and texture (the target is striped), it processes camera images and determines target location in the field-of-view and

[1] The earliest conception of S* in [28] supports additional mechanisms for attention. Their support for designing extensible controllers has not yet been investigated. It is therefore beyond the scope of this paper.

estimates its proximity. Figure 3 describes this in detail.

NextTask. When it observes that the current search task is completed, it selects another search task in the task list.

Rectify. Accepts raw sonar data, smoothes it and corrects occasional sonar failures.

Track. Given location about whether target is to the left, to the right, or not in view, it selects a command to steer left or right in-place. If the target is straight ahead, it selects a command to drive forward. Also determines the "mode" of travel that implements this.

Travel. Determines the system's intended (desired) motion to perform, given a command to perform and a "mode" of travel. It either turns in place or it drives forwards, with the intention of "visiting" a target. Effectively responsible for the robot's displacement.

Verify. Uses the robot's distance to the target to determine if the robot is close to it. If so, it must have succeeded in "visiting" the target.

Representations

Task List. A numbered list of tasks to search for and "visit" some target. Each target is indexed.

Completed Task. A flag indicating the number of the last-completed task.

Database. A database of attributes corresponding to target indices.

Task Info. The attributes of the current target, as well as the current task's number.

Percept. Information about the current target's presence and location in the current field-of-view, as well as a distance estimate to the target.

Ranges. An egocentric representation of distances to boundaries around the robot.

Monocular Camera, ***Encoder***, ***Sonar***. Sensor data from a camera, wheel encoders and a sonar array.

Command, ***Mode***. Command about what direction to move in and a "mode of locomotion" that the *Move* behaviour should use..

Motion Intent. The system's intended (desired) motor action.

Chassis. Robot wheel motors.

The interactions in this behaviour network are as follows. On initialization, all behaviours are triggered once. The *Task List* representation contains a list of targets to "visit" in order, and the other representations contain null values. The representations *Monocular Camera*, *Encoder* and *Sonar* start being updated with sensor information at regular intervals by the framework. The sonar and encoder information is updated every 20 ms and the camera image is updated every 100 ms. *Perceive* does nothing, since it has no task information to operate on, but *Rectify* performs smoothing and other processing to the raw sonar data to compute the radial distances of boundaries (obstacles) in the robot's environment, which is stored in *Ranges*. The update to *Ranges* triggers *Move* to issue a command to the wheel motors, via the *Chassis* representation. This is a null command, initially, since *Motion Intent* is set to null. However, this behaviour is now triggered every 20 ms. Because *NextTask* is triggered, a new task (with number *i*) is pulled off the *Task List*, fetches relevant texture attributes from the *Database*, and stores it in *Task Info* (it will do this again when *Completed Task* contains the number *i*). This causes *Perceive* to filter the camera image in the manner depicted in Figure 3. The resulting information is stored in *Percept*, which is used by *Track* to maneuver the robot in such a way that it either searches for the target with the desired texture attributes or it drives towards it. The *Verify* behaviour monitors the target distance estimate computed by *Perceive* and stored in *Percept*. When the distance is less than a threshold value, the target is deemed to have been "visited", and *Verify* writes the number of the current task into the *Completed Task* representation. This triggers *NextTask* to move on to the next target.

Some observations are in order. Firstly, this is different from the hybrid or purely behaviour-based approach to control. At first glance, this behaviour network may appear to be similar to a hybrid controller, with highly reactive behaviours performing signal processing at the "bottom" of the architecture and the symbol-

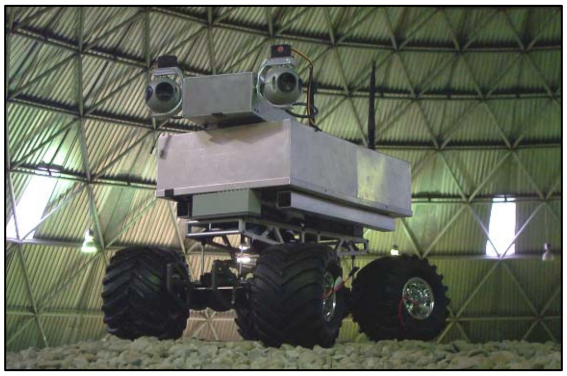

Figure 2: The Argo Rover, the robot used as a testbed [10].

oriented processing (however simple it is) at the "top". However, suppose we were to augment the controller to support path-planning. In this controller, the path planning functionality would belong in the *Travel* behaviour, since it is concerned with displacement. If this were a hybrid controller, path-planning would be closer to the module responsible for task execution, which just happens to be the *Next Task* behaviour, here; but, clearly, *Next Task* is not concerned with path planning. We think of the behaviours and representations that use some explicit representation of the task or of attributes of the task as *task-aware*, because they need to know something about the task in order to do its job (even if it is only the task number *i*). Note that other behaviours, such as *Track*, *Rectify*, *Travel* and *Move* are not task aware because the knowledge of the current task is not needed.

Secondly, this flexibility allows for better organization of perception. Perceptual processing here is done by the *Rectify* and *Perceive* behaviours. The way *Rectify* is organized in this controller is not unusual as compared with the purely behaviour-based or hybrid approaches, because it provides input to a behaviour responsible for action and has no other interaction from the rest of the controller. This is not true for *Perceive*, since it is task-driven and must receive information from other task-aware components.

V. Extensibility considerations and the S* Software Framework.

While the notion of representation in S* addresses the issue of scalability of behaviour-based controllers from the perspective of cognitive systems, it also addresses the requirement for robot control software to be extensible. To investigate this, we have developed, over several iterations, an object-oriented software framework for S*, which we call the *S* Software Framework*. It is essentially middleware for handling interactions between behaviours through shared representations. The framework allows teams of designers to decide on behaviours and representations and on their interconnections, and the framework handles all interactions, including triggering of behaviours by daemons, representation access, sensor and actuator activity, behaviour network initialization and termination, as well as some other functionality, such as message-passing, that is beyond the scope of this paper. The current implementation is in C++ and can compile and run both on Windows 2000/XP and common Linux distributions. To implement a behaviour or representation, designers simply extend an abstract

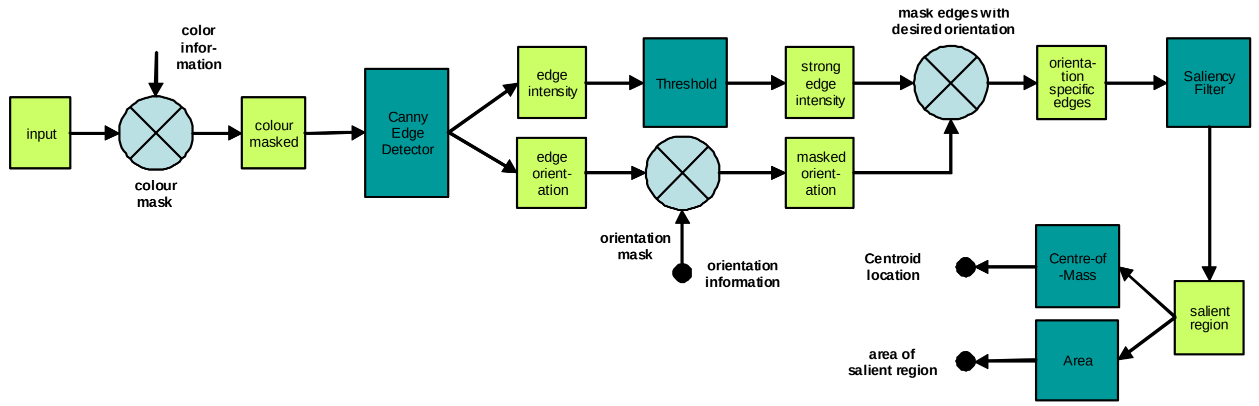

Figure 3: The visual processing involved in the search task for targets by texture. Each target to search for has two attributes: edge orientation and color. They appear above as inputs to the processing stream. An input image (leftmost green square) is filtered to only pass through the target color. An edge detector is applied to produce edge intensities and orientations. The orientations are filtered so as to only pass through the target edge orientation. The resulting data is used to compute information required by the Track behaviour for steering the robot and required by Verify to determine that the target was successfully tracked and that its time to move onto the next task.

behavior or representation class, respectively. In the case of a behaviour, the designer must provide a list of its event and action representations and can also specify a trigger condition. A trigger condition is composed of conjunctions and disjunctions of functions defined over the representations.

That these development activities are all that is required of the development team is why S* controllers are extensible. Loose coupling between S* components makes it easy for designers to add new behaviours and representations to a controller, without being concerned with the implementation details of existing components. A key feature of the framework is that, because behaviour interactions are mediated via representations, behaviours are highly decoupled. Recall that, because representations are abstract data types, they have a public interface and private implementation. A designer of an individual behaviour only needs to know the public interface of one or more S* representations in order to be able to integrate it into the overall system. Keeping a public interface small ensures that behaviours are loosely coupled with their event and action representations, which provides for loose coupling between behaviours. Also, the use of representations to convey information allows behaviours to be asynchronous and operate at different timescales, since behaviours that consume some other behaviours' output do not have to be synchronized with the producer behaviours.

With regards to behaviour reuse, if a designer wants to extend an existing behaviour, he can achieve this easily though the usual inheritance mechanism of C++. Naturally, the same is true with regards to representation reuse. In the particular case of extending the perceptual capabilities of a controller, it is easy to add new perceptual behaviours that employ existing task information representations: design a new behaviour and select them as event representations.

The use of representations does impose one limitation on behaviour design. Behaviours must lock parts of representations for exclusive access when writing to them, which may cause other readers and writers to block. If a behaviour fails to unlock part of a representation, other behaviours could block indefinitely. Because of this, designers must ensure that read and write operations on the same parts representations always terminate (there is no problem if behaviours interact with *different* parts of a representation, however). There is no guarantee that this can be done, short of proving the correctness of the operation's implementation. Operations on representations could possibly fail, although, in our current applications of S*, this has not been a problem for us. In fact, in our implementations, we have applied pre- and post-condition assertions to representation operations, in the style of Design-By-Contract [17]. Any assertion failures that occur have been useful in directing us to the cause of the error.

The focus of robot middleware approaches on integrating functional aspects of intelligent robot control is an important one for robot software. The concern of functionality is orthogonal to the concern of S*, in the sense that designers who apply S* necessarily integrate functional components into their systems, but S* does not prescribe how this should be done. This integration can be realized in a naïve way by implementing the relevant functionality into individual behaviours. S* is not intended to compete with the approaches to middleware mentioned in the Introduction; indeed, they can be instrumental in integrating functional components into an S* controller.

The S* Software Framework has been applied to a number of intelligent agents, both to those in simulation [20], [21], [24] and to mobile robots, one of which was discussed in Section (IV).

VI. Comparisons with Related Work

The idea that attentive processing is important in behaviour-based control architectures was addressed independently by Riekki in his Samba architecture [23]. Samba is a behaviour-based control system augmented with deictics, called *markers*, that select features (from sensor signals) salient to a task for use by behaviours. The markers are effectively an application of task information, because they point to features relevant to the task at hand. The system shifts attention by modifying the markers; this is usually done by other behaviours.

This mechanism limits behaviour input to attended features. Our approach is more general than this: S* does not impose restrictions on how behaviours ought to use information contained within them. Having said this, S* representations are not merely a message-passing interface with triggers. Their semantics are such that behaviours only need to attend to subsets of a representation relevant to the task at-hand, and this is further constrained by the representation's interface definition. This is in contrast to a pub-sub system such as IPC [25], which is merely designed for general transmission of data between distributed modules. Subscriber modules are left to discern what is relevant and to discard the rest in an ad-hoc manner.

Other than Samba, no other architectures are known that are designed to support the task information sharing requirement of perceptual processing.

VII. Conclusions and Future Work

This paper has presented S*, an approach to control that supports task-directed perception through the formal support for explicit representation. We have developed the S* Software Framework that takes advantage of representations in S* to enable designers to build extensible robot control systems. The concerns of S* is orthogonal to functional concerns in intelligent robot control, which are addressed well by current research projects.

The S* Software Framework is a work in-progress, and the current implementation does not address important concerns. Firstly, it cannot run across multiple compute servers having heterogeneous platforms. Secondly, the implementation language of S* behaviours and representations is restricted to C++. Since it has been recognized that current robot control systems must not be limited in this way (e.g. [9], [16]), we are currently developing an implementation of S* that does not suffer these limitations.

In our new design, each compute server maintains a supervisory process that is responsible for executing a subset of the system's behaviours and maintaining representations that are used by them. A novel, service-based distributed computing system, called DataHub, is to be employed. It is based on a publish-subscribe model that operates similarly to IPC [25], except that a service's clients do not depend on a 'central' message broadcaster once they have successfully subscribed to that service. Since robotic hardware devices, such as a motor controller, camera, or pan-tilt unit, will be physically attached to individual machines, it makes sense that behaviours which interact with these devices (indirectly, via representations) should run on the machines to which the devices are connected. For this reason, the supervisors are responsible for executing behaviours on the appropriate machine. As well, since behaviours that share representations might be distributed across multiple machines, the supervisors are also responsible for maintaining up-to-date clones of the representations on the relevant machines. In our new design, the implementation language is not restricted to C++: it will support at least pure C, Java and Tcl/TK.

The next iteration of S* is being developed for PlayBot, our novel, visually-guided smart wheelchair project for assisting differently-abled children at play [22].

Acknowledgments

This research is supported by Precarn Incorporated, the Canada Research Chairs programme, the Canadian Foundation for Innovation and the National Science and Engineering Research Council. The authors thank Prof. Gabriel D'Eleuterio, of the University of Toronto Institute for Aerospace Studies, for use of the Argo rover test bed, and the reviewers for providing valuable comments.

References

[1] Arkin, R.. "Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior," *Proc. ICRA*, 1987. Raleigh, NC., pp. 264—271.

[2] Arkin, R., *Behavior-Based Robotics*, MIT Press, 1998.

[3] Kitano, H., *ed. Robocup-97: Robot Soccer World Cup I*, Springer, 1999.

[4] Asada M., Kitano, H., *ed's. Robocup-98: Cooperative Team Play Based on Communication*, Springer, 1999.

[5] Brooks, R., "A layered intelligent control system for a mobile robot." *IEEE J. of Rob. & Autom*. RA-2, 1986. pp. 14—23.

[6] Brooks, R., (1991a) "Intelligence without representation." *Artificial Intelligence* 47, pp. 139—159.

[7] Brooks, R., (1991b) "Intelligence without reason." *A.I. Memo No, 1293*, MIT A.I. Laboratory.

[8] Bruyninckx, H. "Open robot control software: the OROCOS project", *Proc ICRA 2001*, Vol 3, pp. 2523—2528.

[9] Coté, C., Brosseau, Y., Létourneau, D., Raïevsky, C., Michaud, F., "Robotic Software Integration Using MARIE", *Int. J. Adv, Rob. Sys.*, 3 (1), 2006. pp. 23—30.

[10] Earon, E.J.P., Barfoot, T.D., D'Eleuterio, G.M.T., "Development of a Multiagent Robotic System with Application to Space Exploration", *Advanced Intelligent Mechatronics*, Como, Italy, 8-11 July, 2001.

[11] Firby, R., J., "Adaptive Execution in Complex Dynamic Worlds", Ph.D. Thesis, Yale University, 1989.

[12] Gat, E., "On Three-Layer Architectures", in D. Kortencamp *et al.*, eds, A.I. and Mobile Robots, AAAI Press, 1998.

[13] Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M., "The Belief-Desire-Intention Model of Agency", In J. P. Muller, M. Singh, and A. Rao, eds. *Intelligent Agents V*, Springer-Verlag Lecture Notes in AI Volume 1365, March 1999.

[14] Hexmoor, H., Kortenkamp, D., Arkin, R., Bonasso, P., Musliner, D., eds. Working Notes, *Lessons Learned from Implemented Software Architectures for Physical Agents*, AAAI Spring Symposium Series, March, Stanford, CA.

[15] Kirsch, D., "Today the earwig, tomorrow man." *Artificial Intelligence* 47, 1991. pp. 161—184.

[16] Metta, G., Fitzpatrick, P., Natale, L., "YARP: Yet Another Robot Platform." *Int. J. Adv, Rob. Sys.*, 3 (1), 2006. pp. 43—48.

[17] Meyer, B. *Object-Oriented Software Construction.* Prentice-Hall, 1997.

[18] Microsoft Developer Network, "Microsoft Robotics Studio", Microsoft, Inc., Redmond, WA. web page: http:// msdn. microsoft. com/ robotics/. Accessed Dec 12, 2006.

[19] Nesnas, I.A.D., Simmons, R., Gaines, D., Kunz, C., Diaz-Calderon, A., Estlin, T., Madison, R., Guineau J., McHenry, M., Shu, I., Apfelbaum, D. "CLARAty: Challenges and Steps Toward Reusable Robotic Software", *Int. J. Adv, Rob. Sys.*, 3 (1), 2006. pp. 23—30.

[20] Rotenstein, A.M., "Supporting Deliberation Within Behaviour-Based Systems", M.Sc. Thesis, Department of Computer Science, York University, 2003.

[21] Rotenstein, A. M., Rothenstein, A.L., "Mission-Directed Behavior-Based Robots for Planetary Exploration", ISAIRAS-03, Nara, Japan, 2003.

[22] Rotenstein, A.M., Jacob. D. T., Robinson, M., Shubina, K., Zhu, Y., Tsotsos, J.K. "Towards the dream of an intelligent, visually-guided wheelchair." Proc. *International Conference on Technology and Aging,* 2007. *To appear*.

[23] Riekki, J. "Reactive Task-Execution of a Mobile Robot" Doctoral Dissertation, *Acta Univ Oul* C 105, 1998.

[24] Rothenstein, A.L., "A Mission-Plan Specification Language for Behaviour-Based Robots", M.Sc. Thesis, Department of Computer Science, University of Toronto, 2002.

[25] Simmons, R., Apfelbaum, D., "A Task Description Language for Robot Control." *Proc. ICRA* 1998, Victoria, B.C.

[26] Tsotsos, J.K., "The Complexity of Perceptual Search Tasks." *Proc. Intl. Joint Conf. on A.I., Detroit*, 1989. pp. 1409—1416.

[27] Tsotsos, J.K., "On Behaviourist Intelligence and the Scaling Problem," *Artificial Intelligence* 75, 1995. pp. 135—160.

[28] Tsotsos, J.K., "Intelligent Control for Perceptually Attentive Agents: The S* Proposal" *J. Rob. & Auton. Sys.* 21, 1997. pp. 5—21.

[29] Verma, V., Jónsson, A., Simmons, R., Estlin, T., Levinson, R., "Survey of Command Execution Systems for NASA Spacecraft and Robots", International Conference on Automated Planning and Scheduling, Monterey, CA, 2005.

[30] Utz, H., Sablatnog, S., Enderle, S., Krätzschmar, G., "Miro – middleware for mobile robot applications." *IEEE Trans. Rob. Autom.*. 18 (4) *August*, 2002.