Perception, Action and Effective Representation in Multi-Layered Systems

Glenn S. Wasson, Gabriel J. Ferrer and Worthy N. Martin
Computer Science Department, University of Virginia
{wasson | ferrer | martin}@virginia.edu

1. Abstract

Many robot control architectures possess several layers which view the activity of the agent from different perspectives. The lowest layer typically contains the sensor and effector sub-systems. We will discuss information flow in an agent architecture with an emphasis on structures which facilitate effective perception and action. We start by defining necessary properties of the information flow mechanism. We then describe how information flow in our agent meets the specified criteria. Finally, we describe an implementation of an architecture using these information flow structures on our autonomous agent, Bruce.

2. Introduction

Autonomous mobile agents must perceive and act in the real world. We are interested in vision-based agents, so to understand what agents must do, we need definitions of vision and action. According to Marr, *vision* is "to know what is where by looking" [10]. We define *action* as altering the world in a perceptible fashion. Mobile robots need both vision and action to be closely interrelated if they are to be effective. We will refer to the parts of the control architecture that compose these interrelated functions as the perception/action (PA) layer.

Robots intended to operate in dynamic environments are typically designed with (mostly) stateless sensor and effector sub-systems [5][11]. However, the limitations of such systems in complex domains are well known [18]. We contend that, in order to be feasible, perception needs to take into account the actions which an agent can perform. Similarly, actions need to know about the sorts of results available from perception. Further, the way in which perception and action are bound together heavily depends on the current task of the agent. This close connection necessitates some form of state. We propose a different perception/action layer which uses task dependent structures called markers [4][1].

Any perception/action layer is necessarily limited in its view of the world and the amount of state it can create and maintain effectively. In order to accomplish more complex tasks, many researchers have developed heterogeneous architectures which interface higher-level planning and sequencing capabilities to PA systems [3][6][8]. The question is then, if such higher levels maintain a more complete world model, what information should be passed to the PA system in order for it to perceive/act effectively?

We will argue that any such information flow should be highly structured and minimalist (not provably minimal, but the PA system should not be overburdened). We first discuss the properties that any information flow mechanism between a PA system and the rest of an agent architecture must have to be effective. We then present our PA system and describe how the information flow into and out of it meets these requirements. Finally, we describe a task for which our PA system performs as the lowest layer of a three-level architecture (TLA).

3. Information Flow

We seek to characterize the flow of information within, as well as, into and out of a perception/action layer. The perception/action layer of an architecture is the layer closest to the hardware. Information flows into the PA layer from the higher layers of the architecture and from the environment via sensors. Information from the higher layers configures the PA system for various tasks, while information from the sensors guides the PA system's actions. Many PA layers [7][8][20][21] contain concurrent behaviors whose interaction is an important part of the agent's information flow. Data derived from the PA system's sensors flows out of the PA system to the higher layers of the architecture. This is similar to "supervenience" [14].

The flow of information in a PA system is heavily tied to the internal representation used by that system. All information coming from higher layers of an agent architecture must be converted to structures which the PA layer can use, while all out-going information must be synthesized from

these structures. The internal PA layer representation also serves to coordinate its activities such that a behavior "emerges". In defining the properties of information flow, we are defining the characteristics of this internal representation.

The information flow mechanism (and hence the PA layer's representation) must possess three properties. First, the transfer of information between layers must allow for task-directed behavior. The downward flow must be able to describe goals in some form, but also specify queries to the PA layer about the state of the world with regard to the current goals. The resulting upward flow must be suitable for run-time planning, such as choosing a branch in a conditional plan, or generating a new plan in response to some situation.

Second, the information must be in a direct and accessible form for the PA layer. Since the PA layer interacts with the agent's environment, it needs simple, direct representation to achieve timely performance. The PA layer should not be performing extensive inference on an elaborate world model to select its actions.

Finally, the information transmitted by the higher layer(s) should be as close as practical to the minimum amount necessary to allow the PA layer to accomplish the desired task. While no guarantees of minimality need to be made, the PA layer should not be expected to maintain large quantities of information. For example, if the agent's goal is to go down a hallway, the perception/action layer should receive something closer to specifications of the distance to maintain to the left and right walls, than to a detailed free-space map of the hallway. The perception/action layer cannot afford to be bogged down with extraneous information.

Taken together, the above considerations lead to the idea that the flow of information should be in a highly structured form, i.e. through a well-defined interface. This not only simplifies programming, but leads the designer away from the temptation of creating a PA layer which needs to process arbitrary data structures.

4. A Perception/Action Layer

Perception/action layers are characterized by their "direct" connection between sensors and effectors. This does not imply that there must be a physical connection, or even that there must be a simple table lookup indexed by sensor values [13]. Typically, much of the work in a PA system is in converting the sensory input into a (usually simplistic) representation of the situation (which may or may not be stored) and using it to select one of a few possible actions. For example, to pick up a soda can, the agent must locate the can (using some segmentation and recognition) and then adjust its arm position. The locating of the can occurs on each "loop" of the behavior, regardless of the state of the last arm movement. While these two steps of computing information from sensors and sending commands to effectors take place on each loop of the pick-up behavior, the sending of commands to the arm takes only a small fraction of the time taken to locate the can.

The PA system of an autonomous agent, particularly a vision-based agent, must be efficient in its sensory processing, as that consumes the majority of its time. While this discussion is applicable to an agent with any sensory modality, we will constrain our discussion to vision because it presents the most computational difficulties.

So, given that computational efficiency is of primary importance, and that our PA system will be using vision, the question becomes, what information is needed to effectively complete the tasks specified by the higher layers? For the answer, we fall back to Marr's definition of vision. The PA system needs to know the 'what' and the 'where' for the objects which will fulfill certain roles in the task it is directed to accomplish, *by looking*. The higher layers of the architecture will need to be able to query the PA system for information about the world. To structure this information flow, we use markers [4][1].

For the PA system, performing a task involves enabling a set of concurrent processes, out of which arises the overall behavior of the agent. The flow of markers into the PA system enables various PA processes. The flow of markers out of a PA process (either to other PA processes or to higher layers of the architecture) carries the information which that process has derived.

4.1. Markers

Ullman [19] uses the term "marking" to refer to remembering a location for later reference. He proposes the creation of a "marking map" which holds context dependent locations, in the visual field, that have been analyzed. Attneave and Farrar [2]) suggest that locations outside the visual field can be marked. Pylyshyn describes a similar concept in his FINST model [12]. He describes a limited number of "reference tokens" which can be bound to visual features. This binding is a pre-requisite to determining relational properties about those features. Agre and Chapman [1] use markers in their Pengi system to identify objects relevant to the penguin's current task, e.g. the-ice-block-blocking-my-escape. Brill considers issues involved in using markers in dynamic 3D domains involving occlusion [4].

Markers allow the PA system to maintain a small, highly structured model consisting of task-relevant objects. Each marker contains the `what` and the `where` for some object in the current task. The `what` component identifies the object's role in the current task. For example, the sit-down

task requires some object to fulfill the role of seat. A chair in our lab may be associated with a **seat** marker during the sit-down task, but the same chair may be associated with an **obstacle** marker if it blocks the agent's path to its goal during a navigate task. Note that the `what` component does not have to uniquely identify or even globally classify the associated object. The **`seat`** marker does not say which of all known seats we are dealing with, while the **`obstacle`** marker does not care that the associated object is actually a chair. The `where` component of a marker represents the location of its associated object in ego-centric coordinates. The agent's visual system has primitive tracking capabilities to keep the position of the object stored in each marker up to date. Throughout the rest of this paper we will use boldface to denote values of the various components of markers and phrases such as **seat** marker to denote a marker whose `what` component is **seat**.

Markers also provide the actions to take on their associated objects (goto, look-at, none, etc.). In this way, each marker specifies a "primitive" goal to the action system. For example, a marker whose `what` is **my-opponent** and whose `action` is **chase** represents the goal **`chase my-opponent`**. The **my-opponent** marker provides the location of *the* target which is then used to navigate the agent. Markers may exist to provide the state necessary to some other action, or to express a dependency between objects and goals. For example, an object is only an obstacle if it blocks the agent's path to some goal. If the agent's goal changes, the object may no longer be an obstacle.

Together, the current markers in the PA system represent the aspects of the current task, which are considered important. But, considered important by whom? How does the PA system know what to mark? The answer is that the higher layers of the architecture specify what aspects of the world the PA system needs to consider, and the actions that should be taken. In this manner, the complexity of an agent's overall plan is hidden from the lowest level of the system, but the agent can still preform complex tasks.

The `action` component of the markers specify which behaviors should be activated in the PA system. The `what` and `where` components provide these behaviors with configuration data. So, the PA system can be reconfigured to perform different tasks by passing it a different set of markers.

Our PA system reacts via actions invoked in its hardware, based on the state of its markers and the current values on its sensors. Details on the instantiation and maintenance of markers is provided in a later section. For now, we will concentrate on why the marker structure possesses the desired information flow properties specified above, and why it is an effective structure for execution.

4.2. Flow in the PA Layer

Now that we have defined our PA layer, we will examine how such a system meets the requirements of our information flow mechanism. For purposes of this discussion, we refer to all non-PA system layers in the architecture as the *deliberator*. Later, we will see that the deliberator consists of multiple levels, but for now, this is an unimportant detail.

Information flow within the PA layer and between the PA system and the deliberator displays the required properties. First, markers from the deliberator provide for goal directed behavior because they specify goals to achieve **`(goto the-box)`** or information to obtain **`(switched-on? the-lamp)`**. Queries such as **`(switched-on? the-lamp)`** return both the marker for **`the-lamp`**, and a primitive data type result indicating whether or not the lamp is currently switched on. The state of the lamp's switch is not stored by the PA system as it is not one of the marker components.

Intra-PA layer information flow also provides for goal-directed behavior through relationships between markers. For example, in our system, a **goal** marker with a **goto** action provides the destination for a navigation task. This marker may be provided by the deliberator. The **goto** action consists of four cooperating entities (which we call PA processes). The neck process points the agent's camera toward the goal. The obstacle-detector process creates markers which track objects that represent obstacles to the agent's current straight line path to its goal. The presence of these **obstacle** markers causes another PA process to visually scan the immediate area and create an **intermediate destination** marker indicating a direct path that would cause the agent to avoid the obstacle. Finally, the navigator process heads for the goal, or the intermediate destination, if one exists. The navigator will periodically delete any intermediate destination marker and head for the goal again. If there are still obstacles, they will be re-detected by the obstacle detector, and the avoidance process will repeat itself.

Markers allow the deliberator to communicate its expectations or beliefs to the PA system. The deliberator can, for example, derive an expected position of some landmark, based on its knowledge of the agent's current location and a map of the environment. Passing this "uninstantiated" marker to the PA system will cause the PA system to try and locate a corresponding object in its input stream, thereby instantiating the marker. The uninstantiated marker provides the PA system with the deliberator's knowledge of the expected location of some object. The PA system, in turn, adjusts the estimate by finding the object in the world. The new position can be passed back to the deliberator to refine its map or notion of the agent's position. In this way, the deliberator receives, through the marker, the PA layer's

knowledge (the latest sensor information) about the object.

Markers can be generated by the deliberator because they do not specify arbitrary information about an object. The deliberator layer has access to a database of identification and action routines, along with some form of map of the agent's environment. So, the deliberator layer can construct the information necessary for the PA layer by simply filling in a marker structure containing an object's task-dependent identity, location, and the action to take on it. The deliberator also has the knowledge to construct dependencies between markers when they can be known beforehand. Imagine an agent pouring water from a pitcher into a glass. Since both objects are required for the task, we can say both markers will be dependent on the pour task. However, we usually view this as one of the markers having the **pour** action, and the other marker being dependent on the first marker.

The information in a marker is accessible because it deals directly with relevant aspects of the current task. We make the strong statement that much of the work in any action system is converting sensory inputs into the very information contained in the markers, that is, the location and task-dependent identity of objects. For example, a sonar based agent approaching an object may interpret a short reading as the object. The agent may feedback this information into the motors to decelerate and stop gracefully. In any event, the first computation of the approach behavior was to decide that the sonar reading was registering *the* object (which may be done from context rather than an explicit recognition routine) and how far away that object was.

We argue that the `what` and `where` of markers are world aspects which are always needed to select among the types of primitive actions which a low-level PA layer can be expected to execute. The values from the agent's sensory system will almost always be processed into these properties (with other aspects being computed in response to queries to the PA layer). Marker-based representations make this explicit.

We claim that markers provide a minimalist amount of representation necessary to accomplish various goals. While we do not expect system designers to develop provably-minimal representation schemes for tasks, markers provide a simple, well-defined interface that is effective for use by a PA system. We have found that most of the computational load in a behavior is due to the establishment of the marker properties, so this information is important. We leave the "larger meaning" of having various world aspects in some configuration to the higher layers of the architecture. However, the marker properties are the sort of data necessary to set appropriate motor velocities (avoid the object over there), allocate sensor resources (look at the door on the left), move arms (pick up the coke can at 43cm range and 23 deg. azimuth) etc. Since the PA system tracks the objects associated with markers, PA processes which consult markers will get the most recent positions of the associated objects, even if the objects are not currently in the agent's sensory field [4][20][21].

5. An Application

We possess a small mobile robot named, Bruce. His sensors consist of a single camera on a pan/tilt "neck" and a set of bump sensors. Bruce is connected to a workstation and image processor via radio links.

Here, we are programming Bruce to play tag in our laboratory against a human controlled opponent (a toy truck). Tag provides us with a task requiring navigation through a dynamic environment. The game contains planned exploration of the environment while seeking the opponent, as well as chase sequences, where more reactive control is a must.

The rules of the game are simple. The game begins with the human hiding the truck in some location with in the game area. Bruce then uses his map of the game area to seek out the truck. Once the truck is located, Bruce will attempt to tag it. At this point, the human is allowed to try and avoid Bruce. A successful tag occurs when Bruce closes one of his bump sensors against the truck. Bruce wins if he tags the truck before it reaches a pre-defined homebase location.

The architecture we use for this application is a TLA with a spatial planner (SP) at the top, a manager layer called the task executor (TE) in the middle, and the PA layer described previously on the bottom. The overall responsibilities of the various layers are similar to a number of other architectures [3][6][8].

The basic workings of the application are as follows. The spatial planner generates a night-watchman's tour [17] of the lab from its map (see figure 1). The tour and a more abstract map are passed to the TE. The agent's overall task (for which it generated the plan) is to find and tag the truck. Toward this end, the TE must direct the PA layer to perform two tasks: following the generated route and examining the visual field for the truck. If the truck marker is ever detected, the TE changes the task in which the PA layer is engaged. The TE will change the `action` component of the **my-opponent** marker to **chase** and Bruce will try to touch the truck.

Since one of the agent's active tasks is to seek out its opponent (the truck), the TE passes an *uninstantiated* **my-opponent** marker, with no `where` information, to the PA layer. This indicates that the TE has no idea where the opponent will be, but that the PA layer should try and instantiate the marker at all times, i.e. always be looking for its opponent. Later we shall see how the **my-opponent** marker can be associated with the truck. The other active

task is to follow the generated night-watchman's tour, in the hope that the agent's opponent will be spotted somewhere along the route.

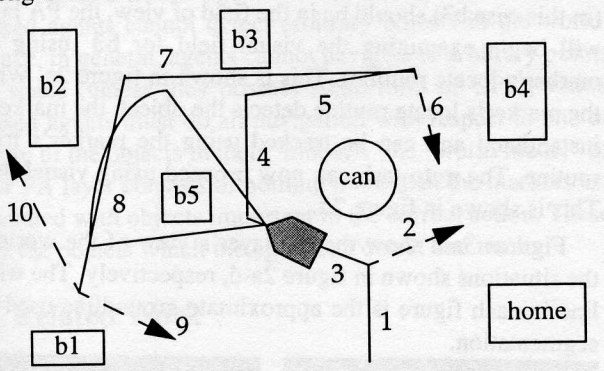

Figure 1. Free-space map and generated plan

Executing the plan generated by the SP involves the TE directing the PA layer to go to various locations. These locations are described to the PA layer in terms of their relationship to objects which the PA layer can identify. Markers for the various objects that the TE expects the agent to encounter along the tour will be passed to the PA layer at appropriate times. In figure 1, for example, step 3 of the tour takes the agent toward box b5. The TE will pass a **goal** marker with a **goto** `action` to the PA system. The goto action will cause the navigation and obstacle avoidance behaviors to become active with the goal of moving the agent toward b5. The PA layer will update the estimated ego-centric position of b5 as the agent moves. When the agent reaches b5, the representation of b5 (the **goal** marker) and any associated **obstacle** markers can be dropped from the PA layer's representation as the **goto goal** task is completed. At this point the agent needs to follow the next leg of its tour, so the TE will pass a new marker to the PA system.

At various points on the tour, the truck-seeking plan indicates that the agent should look around to assure that all the game area has been seen. These points are indicated by the dashed lines in figure 1, with the arrow heads indicating the direction to view. This is accomplished by creating a marker with a **look-at** action and a `where` component in the direction the agent is to look. The **look-at** action will cause the camera to turn to the desired heading, though the PA layer need not associate this marker with any particular object. However, if the truck comes into the agent's visual field via the **look-at** action, the **my-opponent** marker will be instantiated to correspond with it.

An important consideration is how the PA layer's visual system instantiates and maintains markers. In addition to their `what`, `where` and `action` components, markers contain an `identity` component which specifies visual routines used to detect the sensory characteristics of the object with which the marker is to be associated. The association between a marker and an object is initially made by a routine called locate and is maintained by a routine called track. These routines differ from marker to marker, depending upon the associated object's perceptual properties. Note that there is not necessarily a correspondence between a marker's `what` component and its `identity`. For example, when executing step 3 of the plan of figure 1 the PA system contains a **goal** marker whose `identity` is **b5**. However, while executing step 4 the PA system contains a **goal** marker whose `identity` is **b3**. This is because the navigation task requires some object to fulfill the role of destination (goal), but the identity of *that* object can change.

Bruce's visual system uses a "hypothesis and test" strategy. First, the `where` components of all markers are transformed, based on the agent's motion as estimated by data from the wheel encoders, to reflect the current hypothesized ego-centric position of the associated objects. If the estimated location of the associated object should be in the field of view, the agent will attempt to find a correspondent for the marker. The visual system first segments the input images by finding the ground/non-ground boundary, which we call the groundline [9]. Objects are represented by appropriately sized vertical discontinuities in this line (see figure 3). Each image region, indicated by a pair of vertical discontinuities, is analyzed using the track routine of each *instantiated* marker for which the PA layer is seeking a correspondent. Correspondence between the instantiated markers and segmented objects is a function of the results of the track routines (how well a segmented object matches the `identity` component of the marker) and the difference between the location of the object and the `where` component of the marker. If a correspondent is found for a marker, the ego-centric position of the marker is replaced with one based on image coordinates. If no correspondent is found, or the marker's hypothesized position was not visible, the hypothesis becomes the current position. All markers which the PA layer believed should be in view, but were not found, can be dropped as they are no longer present (actually, we don't drop them right away, in order to screen out momentary visual processing failures).

Any unmatched, segmented image regions may be matched against any uninstantiated markers which the PA layer believes should be in view, once the instantiated markers have been processed. This proceeds similarly to the instantiated marker correspondence with the locate routine being used instead of the track routine. When an uninstantiated marker becomes associated with an object, we say the marker is instantiated. The PA layer will now track the object's position and update the location stored in the marker.

Visual correspondence (via the locate and track functions) is based on scaled color histograms [16]. We use 512 bins (3 bits each of red, green and blue). When a track function finds a correspondent, the histogram of that object is

saved to be matched against on the next iteration of the function.

5.1. Searching the Lab

The important concept in this application is that while the "upper" layers of the agent, i.e. the SP and the TE, know about the spatial layout of the game area, the PA layer has only limited knowledge of a specific part of the environment, at any one time. Consider the free-space map of figure 1, which is available to the spatial planner. The SP needs a map of this detail in order to create a night-watchman's tour of the region, allowing the agent to view the entire area in search of its opponent.

However, the task executor need not be bothered with the complexity of the free-space map. The TE needs only the relative distance and orientation between the landmarks which the agent will visit. From this information it can create uninstantiated markers. Consider the agent in figure 2a, which is in the same position as the agent shown in figure 1. As the agent approaches box b5, b5's position is tracked by the instantiated **goal** marker (with `identity` **b5**). This is shown by the solid arrow. In figure 2b, the agent has completed the **goto** b5 task and now the TE is directing the PA layer to **goto** b3. The **goal** marker with (`identity` **b5**) has been dropped, and an uninstantiated **goal** marker (with `identity` **b3**) has been placed in the PA layer's representation by the TE. This is shown by the dashed arrow. Note that the object which the TE wishes the PA layer to associate with this marker is outside the PA layer's current field of view (shown by the lines extending from the front of the agent). The PA layer begins the **goto** task by navigating toward the location in the uninstantiated **goal** marker. This location is updated based on encoder data from the robot. When the marker's `where` component indicates that the object that the TE wishes to have associated with the marker (in this case b3) should be in the field of view, the PA layer will begin examining the visual field for **b3** (using the marker's locate routine). This is shown in figure 2c. When the marker's locate routine detects the object, the marker is instantiated and can be tracked using the marker's track routine. The **goto** task can now proceed using visual data. This is shown in figure 2d.

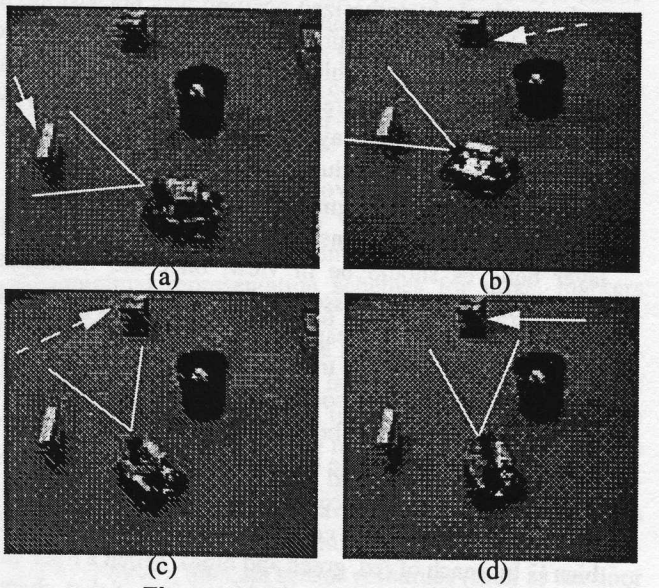

Figure 2. Marker instantiation

Figures 3a-d show the PA layer's view of the world in the situations shown in figure 2a-d, respectively. The white line in each figure is the approximate groundline used for segmentation.

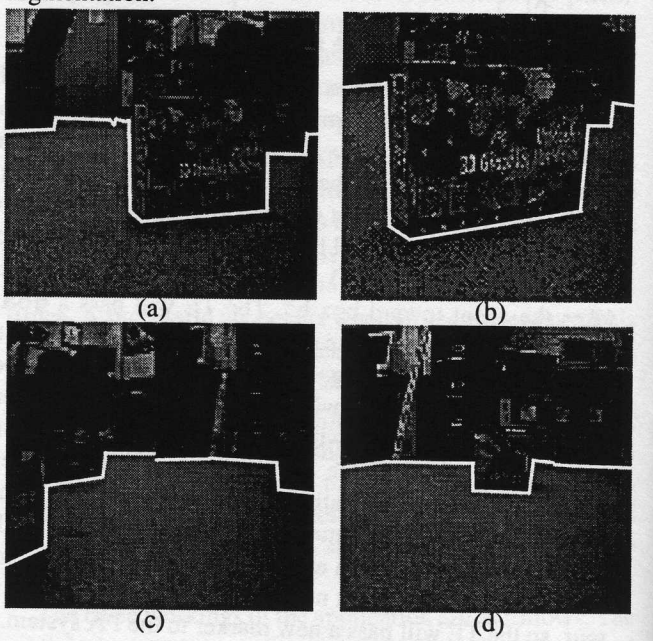

Figure 3. Agent's eye view

The PA layer's computational constraints restrict the amount of information it can maintain and still perform effectively. While some researchers have taken the extreme position of maintaining no information (or state) in the PA layer [5], we feel that small amounts of task dependent state can increase competence at a variety of tasks (see [4] for some examples). The TE need not give the PA system markers for possible obstacles it might meet along the way. In fact, the TE cannot know where these obstacles will be in a dynamic environment. The PA layer does not even represent any non-goal object (for the **goto** task) unless it blocks the agent's path. At that point, the navigator needs the obstacle's position to maneuver appropriately, so the obstacle must be associated with a marker. The agent's vision system, with its marker interface, provides exactly the information needed by the navigator.

Another aspect of markers which is important in the

find-and-tag task is that they provide the PA layer with specific properties that can be effectively extracted from the environment. Unlike agents with proximity sensors, vision-based agents cannot detect arbitrary objects in the world. Since, in general, agents cannot navigate to arbitrary points in space (unless they possess some form of GPS), vision based agents must do all navigation with respect to one or more of the objects in their "model". The "world model" of our PA layer consists of nothing more than the markers associated with objects important to the current action. These are the objects which the agent can detect and track.

6. Related Work

There are many autonomous agent architectures [3][7][8] each having multiple levels of activity. Their lowest layer, often analogous to our PA system, is a collection of independent behavior processes which communicate with each other and the upper layers through some channels [8] or shared memory [7].

Designing these behaviors is mostly an art. While this paper does not alleviate the difficulties of behavioral design, it does seek to say something about the communication between behaviors and the non-PA system layers of an agent's architecture. Work such as Gat and Firby & Slack, allows communication to be unstructured and arbitrary. We have defined a more structured communication mechanism via the passing of markers associated with task-dependent roles. We believe that this interface allows the effective exchange of important information between behaviors or between behaviors and the upper layers of an architecture. Information flow in a marker-based paradigm provides communication of the world properties which a PA system must be expected to routinely compute.

Our work is, in fact, compatible with the work of Gat and Firby and Slack, in that information flow within their PA layers could be marker based and their RAP layers could use markers as the medium for communication with the underlying system.

Spector's supervenience model [14] is similar to our information flow model in that it can be summarized by the phrase "world knowledge up, goals down". The supervenience architecture consists of a set of communicating levels in which lower levels pass facts about the world to higher levels while higher levels pass goals down to lower levels. The first of our information flow characteristics specifies a similar relationship. Each level in the supervenience architecture contains a knowledge base accessible as a blackboard system. Uniform knowledge structures are used at each level. We have presented a knowledge structure, the marker, which is effective for use by a PA system. We further state that the types of knowledge structures needed for the tasks of higher layers are not effective for action at the PA layer. This is because they do not meet the third point of our information flow characterization, i.e. such structures are not meant to contain a practical minimum of information.

The imagination model of Stein [15] involves the creation of an imagination layer under the PA layer. The PA layer interacts with this "imagined" layer as if it were the real world. In this way, Stein's agent which builds a map of its environment as it explores, can be made to map places it has never been by having the imagination level feed the appropriate sensory information to the PA system at the appropriate time. The difficulties with creating an imagination layer for a vision-based agent not withstanding, the idea does present the intriguing possibility of using a planner's expectations to create an imagination layer within which it can "imagine" the outcome of various plans. Our work is compatible with such an idea because we are addressing the flow of information within the agent, while the imagination model address the flow of information from the world into the agent.

7. Conclusions

PA systems have increased task competence over purely stateless systems when given small amounts of highly structured state information. A truly stateless agent cannot even follow a route plan because it cannot store the "next" direction to go. This work goes beyond simply stating that representation can be useful. It provides an organizational structure for PA memory (markers) which can be effectively kept up-to-date. We have also discussed the update mechanism. However, PA systems cannot manage complex tasks. The addition of planning and sequencing components of an agent architecture give the agent new capabilities, but also necessitates some information flow between these components and the perception/action layer. We have discussed a marker-based communication model which provides limited amounts of necessary information (no full world models for example) for effective execution at the PA layer. As part of our TLA, the higher layers allow the PA layer's representation to remain small and task dependent by adding and removing markers as the agent's current task changes. In this way, managing the complexity of the agent's overall goals is lifted from the PA layer, allowing it to be efficient and effective.

We have discussed our PA layer and information flow model in terms of an application in which an autonomous agent searches our lab for an object (a toy tuck) and attempts to actively tag it as it tries to escape. The use of markers allows the layers with map access to provide intermediate goals to the PA layer. The PA layer can also signal

important information to the TE, such as the location of the truck. In addition, markers are used to communicate between PA processes for obstacle avoidance.

We believe that markers provide for a structured communication mechanism between layers of an agent architecture. They describe the properties that low-level systems need to act and can efficiently extract. This is particularly important for designing vision-based agents operating in dynamic environments.

8. References

[1] Agre, P.E.; and Chapman, D. 1987. Pengi: An Implementation of a Theory of Activity. *AAAI-87*: 268-272.

[2] Attneave, F.; and Farrar, P. 1977. The Visual World Behind the Head, *American Journal of Psychology* 90(4): 549-563.

[3] Bonasso, R.P.; and Kortenkamp, D. 1995. Characterizing an Architecture for Intelligent, Reactive Agents. *AAAI Spring Symposium.*

[4] Brill, F.Z. 1996. Representation of Local Space in Perception/Action Systems: Behaving Appropriately in Difficult Situations. Ph.D. Dissertation, Department of Computer Science, University of Virginia.

[5] Brooks, R.A. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14-23.

[6] Firby, R.J. 1987. An Investigation into Reactive Planning in Complex Domains. *AAAI-87*: 202-206.

[7] Firby, R.J.; and Slack, M.G. 1995. Task Execution: Interfacing to Reactive Skill Networks. *AAAI Spring Symposium on Software Architectures.*

[8] Gat, E. 1992. Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots. *AAAI-92*: 809-815.

[9] Horswill, I. 1993. Polly: A Vision-Based Artificial Agent. *AAAI-93*: 824-829.

[10] Marr, D. 1982. *Vision.* W.H. Freeman and Company. San Francisco, California.

[11] Payton, W.P.; Rosenblatt, K.; and Keirsey, D.M. 1990. Plan Guided Reaction. *IEEE Transactions on Systems, Man and Cybernetics* 20(6): 1370-1382.

[12] Pylyshyn, Z.W.; and Storm R.W. 1988. Tracking Multiple Independent Targets: Evidence for a Parallel Tracking Mechanism. *Spatial Vision* 3(3):179-197.

[13] Schoppers, M. 1987. Universal Plans for Reactive Robots in Unpredictable Environments. *AAAI-87*: 1039-1046.

[14] Spector, L. 1992. Planning and Reacting Across Supervenient Levels of Representation. *International Journal of Intelligent and Cooperative Information Systems* 1(3 & 4): 411-449.

[15] Stein, L. 1995. Imagination and Situated Cognition. in *Android Epistemology.* K. M. Ford, C. Glymour, and P. J. Hayes, eds. AAAI Press/MIT Press. p 167-182.

[16] Swain M.J.; and Ballard D.H. 1991. Color Indexing. *International Journal of Computer Vision.* 7(1):11-32.

[17] Taylor, C.J.; and Kriegman, D.J. 1994. Vision-Based Motion Planning and Exploration Algorithms for Mobile Robots. *Workshop on the Algorithmic Foundation of Robotics.*

[18] Tsotsos, J.K. 1995. Behaviorist Intelligence and the Scaling Problem. *Artificial Intelligence* 75: 135-160.

[19] Ullman, S. 1984. Visual Routines. *Cognition* 18:97-159.

[20] Wasson, G.S.; Ferrer, G.J. and Martin W.N. 1997. Hide and Seek: Effective Use of Memory in Perception/Action Systems. *First International Conference on Autonomous Agents*: 492-493.

[21] Wasson, G.S.; Ferrer, G.J. and Martin W.N. 1997. Systems for Perception/Action and Effective Representation. *FLAIRS-97*: to appear.

[22] Wasson, G.S.; and Martin W.N. 1996. Integration and Action in Perception/Action Systems with Access to Non-local Space Information. *AAAI-96 Workshop on Planning, Action and Control: Bridging the Gap.*