Chapter 2  

Related Work

In this section I analyze previous work in both agent representation systems and agent design methodologies. Few agents are designed without some methodological guidance. However, most current methodologies do not explicitly address the problem of representation design and so the representation systems of the corresponding implemented agents themselves must be examined.

2.1. Perception and Representation

The main goal of this work is the design of representations of the environment that are effective for the action oriented portion of an agent architecture. Action in complex, dynamic worlds relies a great deal on perception to provide timely information about the environment. As such, an agent’s representation must be closely tied to its perception. In other words, representation must be continually modified based on perceptual information. Many researchers have developed agents that use representation in their perceptual system.

The Pengi system of Agre and Chapman [2] plays a video game (Pengo) in which the simulated agent (a penguin) does battle with some enemies (bees) in a rectilinear maze of ice blocks. The agent uses data structures known as markers [76]. These structures are associated with particular elements of the video game that are currently of importance to the agent (when running away from a bee, you need to look ahead for the-block-that-is-blocking-my-route to determine if you can escape). These markers are similar to the representa-
tion used by the agents of chapters 4 - 6 in that they store both the position and task function of objects in the environment. Task function is important because the same ice block could be the-block-that-is-blocking-my-route in one task and the-projectile-to-kick-at-the-bee-near-me in another. Later, Chapman’s Sonja system [19] played a more complex video game with fewer constraints on the agent’s actions.

In both Pengi and Sonja, markers are used to bridge the gap between early vision and action. Markers hold locations of important entities in the environment and based on spatial locations of markers, the agent can determine its next action. However, these markers are only a limited form of representation because they only hold locations within the agent’s two-dimensional omniscient (overhead) view. No memory is used to store important locations outside the current view or to allow markers to persist between perception/action cycles. In addition, neither system addresses the perceptual problems of early vision. While Sonja does compute early visual properties to determine locations for markers (Pengi directly accesses the video game’s internal data structures), it does not handle problems with physical sensors, such as a limited field of view, first-person perspective, occlusion and the unconstrained nature of the vision problem.

Kuniyoshi et. al. [45] have developed an agent that also uses data structures, which they call markers, to hold positions of task-dependent points in space. This agent operates in the real world and uses stereo and optical flow to update positions of markers. However, in [65], Riekki and Kuniyoshi state that markers must be associated with perceivable features and so they cannot move outside the field of view. In addition, a single marker in this system can store positions of multiple points in space (markers can specify paths for example) and as we’ll discuss in chapter 7, this can make them difficult to update.
Brill [14] has created a simulated agent that operates in a 3-D, first person domain. This agent represents important task dependent objects in its environment with markers that are similar to those used by Pengi. These markers can, however, represent information outside the current field of view. This expanded area, called the “effective field of view” [13], can be used by a set of behaviors to choose the agent’s next action. The term “effective field of view” denotes the area of the environment that the agent can access in its decision making process. It includes the area currently within range of the agent’s sensors and certain, recent, task-dependent percepts that are stored in the agent’s internal representation. This stored data is treated as if it were another sensor and so the agent can continue to operate in a “reactive” fashion, even though it may be selecting actions based on stored data [13]. Brill’s agent uses a system of representation that is similar to portions of the ones used by the agents in chapters 4 - 6. However, I have expanded the concept of markers to make them effective for inter-layer communication by defining the concept of marker instantiation and by separating concepts that were entwined in Brill’s system. For example, the perceptual properties of objects associated with Brill’s markers were determined by the object’s role in the task (so all food was red and all red objects were food). Brill’s work addressed the issue of whether representation could improve the performance of a perception/action system while this work addresses the issue of the design of representation systems.

Horswill [32][33] has created agents that associate unary predicates with visual trackers (to determine the truth value of some predicate, look at the position of the tracker). These trackers can be inside or outside of the field of view (and are updated correspondingly by odometry or vision). However, while his agents avoid the scaling problems inherent in an Agre and Chapman like system (see section 2.6) by limiting the number of trackers,
Horswill provides no theory or methodology for the design of a system to control their allocation. In chapter 3, I discuss the design of multi-layered representation systems with this goal in mind.

Ballard [8] discusses agents with animated (foveated) vision systems. These agents store the coordinates of important objects that have previously been foveated by their visual systems. Ballard advocates storing the coordinates of these objects in object-centered coordinates, that is, relative to other objects. In this way, the agent only needs to maintain a transform from its current foveal position to an object since the stored object-to-object transforms can be used to transfer the fovea to any other important entity. While this may be more efficient in some cases than the ego-centric coordinate system proposed in chapter 3 (since every ego-centric coordinate must be updated when the agent moves), Ballard’s approach is fragile if objects can move relative to each other. This thesis assumes that agents designed by this methodology will be operating in dynamic environments where the object-centered approach will be too brittle and so advocates paying the computational price for an ego-centric representation.

2.2. Representations for Navigation

Maps of an agent’s environment are a common form of spatial representation. Since many tasks for autonomous robots (including those in this thesis) require navigation, maps are quite useful. Grid based maps, called occupancy or certainty grids, have been implemented using sonar [12] [55] and stereo vision [54]. These representations describe the free-space surrounding the robot and so are most suited for navigation and obstacle avoidance. Since these techniques register only occupancy, their representation is not suitable for a general representation of important properties of objects in the environment. Kuipers
[42][43][44] has developed maps for indoor and outdoor “large-scale” environments. His TOUR model defines a semantic hierarchy of information where <view, action> pairs are accumulated to form topological and then metric maps. A <view, action> pair means that doing “action” should put the agent in a position where its sensors perceive “view”.

Both occupancy grids and the TOUR model’s maps (indeed all map representations), represents space and not objects. This means that computational effort must be expended to “extract” object data from the map. Typical tasks for autonomous agents, including navigation, are with respect to some object (e.g. “pick up the book”, “go to the store”, “hit the ball”) and therefore will need object data. A perception/action system should not use map representation as, for any large map, it will be prohibitively expensive to extract object information. However, other layers of the architecture can (and the agents in this thesis do) use maps. Kuipers work is complementary to mine in that even though maps are an inappropriate representation for a PA layer to use in choosing actions, the PA layer can be used to build a map using the TOUR model. The <view, action> pair are the right “granularity” for a PA layer to generate because they correspond to the phases of the PA layer’s execution loop, i.e. perception (view) and action. The synthesis process that builds the maps could be performed by another layer of the architecture that receives these tuples from the PA layer.

Other researchers have also used map representations. Kosaka and Kak [41] have developed an agent that navigates using CAD models of its environment, visually matching model features to perceived features. CAD representations are difficult to obtain (someone has to create the model) and brittle. The brittleness is due to the fact that CAD models model the static properties of the environment, those portions that are thought not to change. However, the agents of interest here exist in domains where dynamic (not static) features
of the environment are important. Miller [53] presents a mapping representation that is based on regions. Each region defines the number of degrees of freedom about which the robot can determine its position based on local sensory information. In an open area, the robot can determine nothing about its position or orientation, while in a dense area of objects, the robot may be able to totally orient itself. Mataric [50] uses a topological map built as a distributed collection of “behaviors”. Activation energy spreads outward from the node at which the robot is located, activating nearby behavior-nodes to begin looking for their associated sensory signatures in the sonar input. The main advantage of her technique is that the map actively maintains itself, adding or removing links between nodes as the agent travels or fails to travel between perceptually distinct areas of the environment. However, these maps are non-metric and so cannot always be used to make efficient path plans (just topologically short ones).

All of these representations are useful for self-localization, i.e. determining one’s position with respect to a map, but are not a good representation for general tasks. This thesis provides a means of breaking down large-scale spatial knowledge, such as that contained in maps, and making it available to the PA layer as computationally affordable chunks of local-space information.

2.3. Agent Architecture Design Methodologies

There are many agent design methodologies in existence today. Most are concerned with dividing the agent’s tasks among various components of the agent’s architecture based on the nature of the tasks. Although most scarcely address the design of representation systems (Brooks [17] being the exception), they do each advocate the use of one or more representation systems.
Early work in the design of agent architectures for dynamic environments was done by Brooks [15][16][17]. He proposed eliminating all representation because it would invariably become inconsistent with the state of the world and therefore lead the agent to make incorrect decisions. While this approach has been rejected as heavy-handed by most researchers (including the Representation subgroup of the 1995 AAAI Spring Symposium: “Lessons Learned from Implemented Software Architectures for Physical Agents”), it did cause a re-evaluation of the role and structure of representation in the perception/action components of agent architectures [14]. Brooks himself used internal state, i.e. representation, in most of his robots [17].

The RAP system [28] has been used in a number of autonomous agents with multi-tiered architectures, most often as part of the 3T architecture [10]. RAPs use lisp-style predicate structures to describe the state of the world. While these have certain advantages (for systems written in lisp) such as the ability to perform unification and their ability to be used as a fact database for a classical planning system, they are fairly unstructured for perception/action systems. Unstructured means there is no limit on the number or content of these predicates. The RAP system was not designed to operate in tight perception/action loops with the environment and so it is not surprising that its representation system is inefficient for such purposes. The original RAP work [27] used a simulated perception system that wrote time-stamped sensor predicates to RAP memory. This required the RAP system to search through its list of facts to determine which were current and required some form of garbage collection. Such searches present an unacceptable time sink for agents in dynamic environments and are the result of trying to make a PA system operate like a planner.

The 3T architecture [10] uses a layer below the RAP system called the skill layer. This
layer sends “event” messages to the RAP system over a low-bandwidth channel, allowing RAP memory to contain smaller numbers of “higher level” facts. For example, when pouring a cup of coffee, instead of storing visual state used to accurately track the coffee pot while aligning it with the cup, the RAP memory can simply hold the fact that the pot “is aligned” with the cup (allowing the RAP engine to tell the skill layer to begin the pouring task). While the accumulation of perception and action information into events reduces the number of predicates and hence the search process of the RAP layer, this dissertation argues that any representation system can be made more efficient if it can be given appropriate structure. Chapter 6 describes an agent that uses structured representation to encode “higher level” properties of the world (such as those stored by the RAP system) without the need to perform unification to derive them from a database of information.

It should be noted, however, that this work does not dismiss the RAP system’s sequencing capabilities or processing engine. In fact, the representation design methodology presented here is complementary to such a system, since RAPs do not address the design of representation and this work does not address the structuring of agent processing capabilities1.

As mentioned previously, the 3T architecture has at its lowest layer, a collection of independent, parallel behaviors called skills. These skills are typically arbitrary C code and can pass arbitrary data structures between themselves and the RAP system. The 3T design philosophy is concerned with deciding which tasks should be handled by which layers of the agent architecture. The design of the processes that carry out those tasks is left to the

1. I do address the structure of an agent architecture in that I believe it should be multi-tiered, as with many other architectures [10][29][5][71]. I also designed sequencing layers for the agents in chapters 4 and 6, but I do not assert that my sequencing layers are superior to the RAP system, merely that they perform similar functions.
agent programmer. This thesis’ methodology proposes that arbitrary data structures are the
wrong representation for PA systems because they can force the PA system to do unaccept-
able amounts of work to sort through the represented data. This slows the PA system’s de-
cision making process. In addition, arbitrary amounts of data can take arbitrary amounts of
time to verify. The PA system can not check if arbitrary representation is consistent with
the world state and remain responsive.

Gat’s ATLANTIS [29] architecture has representation in all its layers. Skills in ATLAN-
TIS’s perception/action layer are written in a language called ALFA [30], which allows in-
ternal state variables. These variables, however, are simple data types (integers, floating
points, etc.) used to implement loops and such, and are not effective to model the environ-
ment. Of course, such data types can be used to encode information about the environment,
but since neither ALFA nor ATLANTIS place any limit or structure on the representation,
the representation has the same deficiency as the arbitrary data structure representation al-
lowed by 3T. Gat’s sequencer layer, which sits above the PA layer, is modeled after Firby’s
RAP system and so uses similar memory structures (with the same benefits and limita-
tions).

An alternative architectural layout is proposed by Simmon’s TCA architecture [69]. His
work is concerned with interleaving planning, sensing and execution through TCA’s plan
representation, a fundamentally different type of representation than that addressed by this
thesis. Although TCA’s modules appear to hold internal state, that state’s construction is
left to the module designer and thus suffers from the potential for arbitrary data structures.

The architecture of Noreils and Chatila [59] is somewhat similar to 3T in its division of
tasks between components of the architecture. While they also allow arbitrary data struc-

tures, they do use a technique similar to my representation instantiation (see chapters 3 through 6). Beliefs about the world are expressed to sensory modules that then try and fit world data to them. The Supervenience architecture [71] has many layers, each using the LISP clause as the fundamental “unit” of information. The layers share a common memory system and have the same problem as Firby’s RAP memory, i.e. it becomes expensive to both find and “forget” (throw away) information among all the predicates.

Albus’s RCS architecture developed at NIST [3] was designed to control many types of mechanical devices, not just mobile robots. It defines a general representation structure used by all levels of an architecture in which higher layers have progressively more powerful inferencing capabilities. Unfortunately, this allows planners and PA layers to use the same representation structures. I believe that the techniques available to planners allow them to use representational structures that are ineffective for perception/action systems. I recommend creating separate (but linked) representations of the world for planners and PA layers to avoid the temptation of using inference in the PA layer itself.

Arkin’s AuRA [5] architecture separates PA layer and planner functionality, but is mainly concerned with navigation at the PA layer. Its representation is geared toward potential fields representing travel paths for the agent. Connell’s SSS architecture [22] has only limited representation of the environment’s geometry in its symbolic layer and none in its subsumption or servo layers. This architecture has only been applied to navigation and so the lack of representation was reasonable. As stated previously, I believe stateless systems are not reasonable for general tasks.

CIRCA [56] is an architecture concerned with making real-time guarantees about its performance. The authors do not address the design of efficient representation systems, but
much like the RAP system, there is no reason why a well designed representation could not be used in this architecture.

Certain extensions of the Soar architecture, such as Robo-Soar [46] or Air-Soar [62], represent cognitive architectures that control agents that interact with the world. Soar is a cognitive architecture that can pursue a number of hierarchical goals simultaneously. The key feature of the Soar architecture that makes it different than other multi-tiered architectures is that each layer in Soar has the same control structure, it just operates in a different problem space [46]. Since I am interested in representation design, it is beyond the scope of this thesis to address the advantages and disadvantages of this architectural structure vs. other layered systems. Instead, I address the use of representation in this architecture.

Soar has a two level memory system, where “working memory” contain knowledge about the agent’s environment and “long term memory” contains productions that contain the control information that the agent uses to select and reason about actions. This thesis is interested in designing representation whose function is akin to “working memory”. There are a number of problems with the memory system in these versions of Soar, such as LISP-predicate style representation and a uniform representation at all layers. These are discussed in more detail in chapter 7.

Uniform representation is used at all layers in Soar because all layers are essentially production systems. The problem is that this means that the information used for long-term planning and reactive in a dynamic environment is contained in the same structure. This can tempt a designer to store information that is useful for planning in a layer concerned with reacting. Without trying to constrain what is contained in the representation of the layers of the Soar systems that interact with the world, the Soar methodology allows these lay-
ers to bog down trying to search through and maintain that information.

2.4. Psychological Representation

There has been considerable work by psychologists on the form and function of representation used by the action oriented portions of the human cognitive architecture. That is, how do humans represent information that is not for the purpose of symbolic, cognitive thought (such as planning), but rather for sensori-motor system use when carrying out actions? Although some studies show that people will not use memory when that option is available [7], none have proposed that humans have no representational system in the manner of Brooks [17]. Particularly in the visual domain, representation plays an important role. Aloimonos has shown how certain ill-posed vision problems (e.g. shape from motion) become well posed when the agent can use multiple camera views with an understanding of the motions between them [4]. This implies some memory of previous images, which itself implies some representation to hold those memories.

Shimon Ullman proposes a model of the “intermediate” human visual system based on small programs called visual routines [76]. An important concept used by these programs is the ability to remember, or “mark”, portions of the image that have already been analyzed. He proposes a “marking map” that holds the location of portions of the scene described by the incremental representations built during scene analysis.

Pylyshyn and Storm [63] propose the FINST model where a limited number of “reference tokens” can be bound to visual features that they then track. Associating a FINST with a visual feature is a prerequisite to further processing involving that feature. The FINST model is an object-based model of attention, as opposed to a location-based model, in that FINSTs do not point to particular locations in space, but to particular visual features (and
continue to do so as the features move). It is different from other attentional models in that it supposes that attention can be directed to multiple places in parallel. This position is further supported by Yantis [85] whose experiments show that people can track multiple independent elements by imagining them as the vertices of a “virtual polygon”. While the previous works have been concerned with targets in the visual field, Attneave and Farrar [6] show that subjects can remember and track the locations of elements that were initially within the visual field, but have since moved out of view. The parallel, object-based model of attention, where targets can be in or out of the visual field, is consistent with the representation systems designed by this thesis.

Other researchers have concentrated on how representations are used for action. Ballard et al. [7] describe how cognition on the scale of actions (approximately 1/3 of a second) consist of small “programs” with “variables” that are bound at action time. These “variables” are bound to objects, parts of objects, or visual features of appropriate world aspects that will be manipulated by the action program.

This thesis advocates the creation of linked layers of representation where the represented information gets more complex (requires more computation to determine) as the layers that use it get further from the PA layer. Psychology researchers have investigated similar links between representations used by different bodily systems. Feldman [25] investigates the connection between the representation used by the visual-motor system and more cognitive representations that might be used for other tasks. He describes a hierarchical representation system with four levels. The first is the representation of space within the current fovea and the second synthesizes various foveal fixations to (ego-centrically) represent the “stable world”. This system, Feldman says, primarily interacts with the “environmental
frame” that encompasses the other two levels and represents space in a non-body coordinate system.

Although not strictly dealing with representation, the two-handed rod manipulation experiments of Guiard and Ferrand [31] show the usefulness of multiple representations. People manipulate rod handled tools (like a rake or a broom) by having one hand perform coarse motion and the other fine motion. The fine motion hand operates in a coordinate system set by the coarse motion hand. While it seems plausible that people can represent a rod handled tool as a single conceptual entity, if they need to manipulate it, representations of the rod in different coordinate systems are needed. In the deictic program paradigm of Ballard [7] variables for right and left hand positions would be bound to portions of the rod, probably with one hand’s position defined in terms of the other’s.

2.5. Other Design Methodologies

Since designing the software for a robot architecture is an exercise in software engineering, methodologies in this field must be examined. There are two principal places in which current software engineering design strategies can be applied: the decomposition of the agent’s goals and the design of an agent’s representation. While the latter is the primary concern of this thesis, the former is a necessary step in the methodology of chapter 3. No current software engineering methodology addresses the design of representation systems for autonomous robots, so I address the use of the methodologies in decomposing the agent’s task and how that may lead to a system of representation.

Structured programming [84] decomposes the problem using a top-down approach. The problem is divided into sub-problems, each of which are further divided until some basic level of implementability is reached. The methodology presented here applies this ap-
approach, as well as a bottom-up approach, to decompose the agent’s goals into a series of tasks. Since structured programming is a tool for general software design, it has little to say about the implementation of the various tasks in the decomposition and so is open to all the problems of using arbitrary data structures for representation.

Object oriented design [11] espouses a philosophy of decomposing software not along functional lines, but into a collection of cooperating entities, called objects, that encapsulate both data and the functions that operate on that data. Object oriented design does not seem to map to the problem of decomposing the agent’s tasks as well as structured programming. Since the decomposition is not by function, the designer has to do something like creating “goal objects” in which the agent’s capabilities represent functions that move the world state (data) toward some goal. These goal entities would then have sub-goal entities that moved portions of the parent goal’s world state toward a goal. While other possible interpretations of the “objects” in the agent’s system are possible, a pure functional decomposition is what is needed in the end. The designer has to reason how to use the agent’s capabilities to achieve its goals and the definition of “objects” here seems to get in the way.

However, unlike other methodologies, the object oriented approach has much to offer in the design of representation systems. The representation systems developed in the following chapters are “object based” where object refers to a relevant aspect of the world (often a physical object). These representations encode both an object’s relevance to a particular task and how a particular action will be carried out on that object. For example, if the agent’s task were to pick up a soda can, the representation would indicate that some object was a soda can, and as such, picking it up means positioning the effector(s) in a certain way. Note that the action of picking up a dumbbell would require different effector positions and
so a representation that indicated an object was a dumbbell would have to encode those. This combination of function and data in a single structure is similar to the object-oriented philosophy. In fact, the encapsulation/information hiding aspect of the object-oriented model is particularly appealing because it maps well to this thesis’ concept of creating “roles” in tasks that can be “filled” by various entities in the environment (see section 3.4). The roles form equivalence classes of objects that can be used by the agent to perform some function. The details of the selected object can be (more or less) hidden from the agent’s action selector. The object-oriented paradigm’s abstraction of class hierarchies is also useful, in the design of levels of representation. If different levels are used by different layers of an agent architecture, different abstractions of the environment can be placed in different levels so that the different layers can operate on data at the right “granularity” for their decision making processes.

The object-oriented model does not, however, point out the issues that most effect the structures that make up the agent’s representation. The methodology presented here directs the designer in considering the structural impact of issues of perception, action and communication.

Communication among the “functional units” (be they objects or functions) in a software design is another important issue. Both data-driven [23] and event-based programming [67] have been proposed to model communication between units. Since communication within the agent architecture is of concern to this thesis, these paradigms can be of use.

Data-driven programming structures a program’s “units” into functional blocks where data flows between them along defined pathways. Units are defined in terms of their inputs and outputs with the internal design left to the designer (possibly using another methodol-
ogy). Programs become data-flow graphs that define order of execution by data availability and have no notion of a system wide program counter. Modeling communicating skill networks this way is appealing because any natural parallelism is exploited. However, data-flow graphs themselves do not say anything about how the data should be structured, or what data should be communicated. My design methodology addresses both these questions.

Another communication paradigm is the event based paradigm. In this paradigm, computational units register with other units to receive notification when certain “events” take place. This programming paradigm is often used in GUI systems to represent such asynchronous events as mouse or key clicks [67]. In this paradigm, behaviors may communicate by having the agent’s perceptual system generate events in response to certain changes in world state. Those changes would be captured in specific data structures (representation) for the event. Any behavior registered to receive notification of a particular event would receive this representation when that event was generated. The agents developed in chapters 4 - 6 have behaviors that communicate in a manner similar to a blackboard system [47] and events could be used as the means of informing behaviors that new data is available on the blackboard. However, this is an implementation detail and does not address the format of the data as this thesis does.

2.6. Scaling Problems

Many researchers have investigated what I will term as the “scaling problem” for action oriented architectures. That is, can these architectures be made to perform complex tasks? The architectures of Brooks [17] and Agre and Chapman [2] differ in their use of state, but both perform no planning and follow no plans. While following no pre-defined instruction
sequence may make sense in some dynamic environments, researchers have questioned the limits of the tasks that such agents can perform.

Kirsh [39] refutes Brooks’ claim that one can substitute control for representation in an agent architecture. In order for this to be true, there must always be enough perceivable (in view and not occluded) cues in the environment that an agent can select an action that will have no adverse consequences down the road. This is the only way that an agent can (ultimately) get away with not considering alternate courses of action, i.e. planning. Kirsh believes that reasoning (planning) is required for complex tasks and planning requires an abstract, symbolic model of the world. Therefore the representations that Brooks seeks to eliminate cannot be removed from the agent architecture entirely.

Tsotsos [75] also argues that the pure Brooksian approach to agent construction cannot scale to human intelligence, by showing that visual search, a common activity in intelligent agents, is NP complete without a target (actually, without an “explicit” target, meaning the agent doesn’t have a description of its target that can be used to simplify the search process). This “unbounded” visual search involves grouping random collections of pixels and analyzing them to see if one of the agent’s stimuli is present (and so the agent should execute its response). Interestingly, Tsotsos also shows that with a small amount of state (a description of the target being sought), visual search is linear. A perceptual description would have to be stored somewhere, i.e. in some representation. The representation designed by this methodology provides targeted visual search.

Pengi [2] has some limited state in its visual system that can limit the complexity of the search task. Agre and Chapman refer to systems with this limited state as deictic, or pointing, because the state “points to” (holds) only currently relevant aspects of the game. How-
ever, as pointed out by Ballard [7], the Pengi architecture is a cognitive one and not a visual one. That is, the advantage of such an architecture is the simplicity of the process of selecting the current action when behavioral “variables” (markers) can be bound to different objects in the game over time. The problem is not with what the agent stores in its internal state, but with what it might have to store at any given moment. The agent must search the entire (perceivable portion of the) environment, all the time, for all the stimuli to which the agent has a response. Even after a marker is bound, it might need to be rebound to a different object, e.g. when the-bee-closest-to-me changes to a different bee. Agre and Chapman had access to their game’s internal data structures, which contained a unique labeling of each object in the game. This allowed them to switch their markers to new targets at little computational expense. However, for systems that must deal with physical sensors, the more complex the domain, the more important aspects there will be to register and thus the more computational burden there will be on the perception system.

Maes [48] creates a behavior-based system with a similar scaling problem. In her architecture, behaviors are connected to other behaviors whose preconditions they achieve. Sensors put activation energy into the behavior network based on precondition predicates they detect as true and when a behavior has all its preconditions activated, it will run (possibly activating other behaviors). For a complex task, there can be huge numbers of predicates that the agent needs to monitor for and since there is no behavior hierarchy, there is no means of limiting the network to considering only those predicates needed by the current task. In other words, the agent must always look out for stimuli related to all tasks that it might ever want to execute because at any time the requisite predicates may become true causing a switch in the agent’s activity. Bryson [18] argues that the complexity of creating,
debugging and tuning such a fully parallel, behavior-based system (as opposed to one with a control hierarchy) outweighs the benefits in reaction time that parallel systems may provide. In her study, the tuning of such a system was so difficult that it could not be made to perform as well as a system with a behavior hierarchy, at the same task.

Neither the stateless nor the purely deictic behavior-based approach scales. As we’ll see in succeeding chapters, my agents attempt to avoid this scaling problem by only looking for aspects of the environment that are important to their current task, even though they lose some ability to recognize serendipitous circumstances because of this.

A related piece of psychological work is presented by Patalano and Seifert [61]. They show that humans can encode known future goals or goals that they have been sidetracked from by associating them with objects required to complete them. If a subject sees some item needed to complete a goal other than the one they’re currently pursuing, they may grab it since they’ll need it soon. This is interesting because it shows the people can recognize objects as useful, even when they are not related to their current activity. While this seems to leave humans open to the same scaling problem, the experiments show that people do not recognize more and more objects as useful as they get more and more goals. They seem to strike a balance between the number of pending goals and the items they recognize as important to achieving them. When there are too many goals, people will focus on one at a time and fail to recognize important objects for pending goals. The methodology proposed in this thesis does not preclude examining the environment for aspects that may be useful in the future. It does however, caution the designer that the computational cost of too much “look ahead” can ruin the reaction time of the agent.