This chapter proposes a design methodology for autonomous agents operating in dynamic environments. The main contribution of this methodology is a structured method of analyzing an agent’s task and capabilities to guide the development of the agent’s representation system. Agent designs based on this methodology are intended to interact with the world in a rapid, stimulus-response fashion. That is, the time delay between perception and action must be as small as possible. The component of the agent architecture dealing with sensors and effectors is called the perception/action (PA) system and is conceptualized as running in a “tight” loop using sensor data to control effectors and effectors to manipulate objects and direct sensors. Due to the computational complexity, the PA system must have almost no explicit inferencing capabilities to remain responsive. Agents designed via this methodology are also assumed to operate in dynamic domains, i.e. environmental change causes perceptions to become outdated.

It must be understood that this is a methodology and not an algorithm for agent software design. As such, the methodology cannot generate provably optimal agents along any lines. In fact, there are no tasks that necessitate the use of this methodology. However, this methodology can be a powerful tool to focus a designer’s attention on certain aspects of “agent-ness” that are of critical importance to the creation of autonomous systems, particularly the representation used within the control structure. I will argue that this methodology can help
designers create agents that are efficient and effective at tasks in the domains of concern defined below.

3.1. Task Domains

As discussed in section 1.3.1, the task domains of interest in this thesis share a number of characteristics. They are summarized in table 3.1.

Table 3.1: Summary of Domain Characteristics

<table>
<thead>
<tr>
<th></th>
<th>The environment is dynamic</th>
<th>Current sensor data is more valid than previous sensor data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>The task contains different aspects.</td>
<td>The different facets of the task can be handled by “independent” architectural components within the agent.</td>
</tr>
<tr>
<td>3</td>
<td>Knowledge of large-scale space [43] is required.</td>
<td>The agent cannot perceive all important aspects of the environment a priori or at any one time.</td>
</tr>
<tr>
<td>4</td>
<td>The task can be decomposed.</td>
<td>The task can be described by a hierarchy of tasks and subtasks and allows important environmental entities to be viewed at different levels of abstraction, e.g. part and whole.</td>
</tr>
<tr>
<td>5</td>
<td>The entire plan cannot be made a priori, or replans do occur.</td>
<td>The task requires the agent to have on-going “higher level” control.</td>
</tr>
<tr>
<td>6</td>
<td>The environment is suitable quantifiable with coarse metrics.</td>
<td>The agent can use crude maps effectively.</td>
</tr>
</tbody>
</table>

Domains with these characteristics are well suited to agents with perception/action systems. The methodology emphasizes aspects of the design process that are important in such domains.

3.2. A Design Methodology

This section outlines a design methodology for autonomous agents. The steps of the methodology take the form of a series of questions that the designer must answer. For each
step, I briefly describe the questions and illustrate them with a running example of an agent that is designed to “walk the dog”. I then discuss the rationale behind the questions and summarize the contributions of the step to the design process.

At the start of the design process, certain facts about the agent, its task, and its environment must be known. For the task and the environment, a high-level specification must exist along with the availability of more detailed information if it becomes necessary. For the agent, the designer needs a description of its capabilities. In the example task, the agent must walk a dog along a planned route to the park (see figure 1) and then play fetch with the dog. For exposition purposes, figure 1 represents a number of constraints on the task. The agent must follow the indicated route, must stay on the sidewalk, must cross the street where indicated, etc. A complete list of constraints appears in appendix C. The agent’s environment consists of a road, sidewalk, crosswalk, and dog. There are also several flower beds along the route to the park and possibly cars in the street. The important agent capa-
bilities include navigation abilities, a leash effector that can be reeled in to keep the dog within various distances, a pressure sensor to detect which direction the dog is pulling the leash, a vision system that can identify important landmarks, curbs, tulips, roses, the dog, the leash, the ball, the mailbox and cars, as well as a reloadable ball launching system so the agent can play fetch with the dog.

Throughout this chapter, I will refer to many agents and their tasks (including the “walk the dog” agent). The task, environment and capabilities of these agents will be summarized in insets like task specification 1. This thesis broadly defines capabilities as anything that the designer has to work with at the beginning of the design process. Capabilities can be anything from detailed hardware descriptions like a stepper-motor to full behaviors like “navigate to a landmark”. For the “walk the dog” example, agent capabilities are mostly given as behaviors (e.g., a navigation system that can maneuver to landmarks, instead of wheels and a camera), but for other agents, bottom-up thinking is applied to turn capability descriptions into behaviors (see chapters 4 - 6).

Finally, I use hyphenated phrases, e.g., walk-the-dog, as proper nouns to aid the reader
in understanding the goal of a task. While these phrases appear to make certain steps of the methodology trivial, they are merely names that could be stripped of their semantic content by using titles like task-37. However, it is easier to understand that the goal of the walk-the-dog task is to have the dog walked, as opposed to the goal of task-37 being to have the dog walked.

3.3. Task Decomposition

The first step in the methodology involves creating a hierarchical decomposition of the agent’s tasks. The agent will have certain capabilities (both hardware and software) that are given to the designer. These can be thought of as the agent’s “primitive” skills, i.e. the fundamental processes from which all of the agent’s actions are built (see section 1.3.2). If the task decomposition is thought of as a tree with the original task specification at the root, these skills are the leaves. The job of the designer is to bridge the gap between the task specification and the skills. When creating the decomposition, it is important to consider the following:

*What are the agent’s primitive skills?* The answer to this question depends on the agent’s capabilities and the degree to which each can be viewed as a “black box”. Although the design begins by considering the skills, ultimately creating a task decomposition must be both a top-down and bottom-up process. The designer must decide on the agent’s skills while breaking down the overall task into a sequence of subtasks that achieve the agent’s goals. This brings up several more important questions.

*Which tasks can be decomposed into sequential subtasks? Which can be decomposed into*

---

1. Task decomposition is a major part of general requirements analysis and it remains mostly an art [66][84]. This methodology provides a structure in which to practice that art.
parallel subtasks? Interior nodes of the decomposition tree will be broken into subtasks whose overall outcome is meant to complete the parent task. In this way, the leaf skills can be combined to achieve a greater range of behavior than that of the individual skills.

3.3.1 Example Task Decomposition

In figure 1, we can see that the dog must be taken along a certain route to the park (remember task specification 1 and figure 1 represent the given task constraints). First the dog must be walked along the sidewalk to the street corner with the crosswalk. Then the agent and dog must cross the street on the crosswalk and go into the park. Finally, the task specification says the agent should play fetch with the dog. We can derive further specifications for the steps on the way to the park by examining the map in figure 1. For example, to keep the dog on the sidewalk, the dog must be kept out of the street and out of the flowerbeds. At the crosswalk, before crossing the street, the agent should look for oncoming cars.

A few more details can be discerned by careful thought about the processes involved. When walking the dog, the agent moves and the dog is meant to follow. If the dog does not follow, the agent must tug on the leash. Since the leash is the agent’s only mechanism for controlling the dog, the leash must be manipulated to keep the dog “safe”, i.e. on the sidewalk, and to keep the dog moving forward. A good strategy for controlling the dog might be to keep it nearby and just tug the leash when the dog’s heading needs to be changed, but in the end this is up to the designer.

When the agent is in the park, it needs to play fetch with the dog by repeating the cycle of throwing the ball and waiting for the dog to retrieve it. However, the game should come to an end when the dog has had enough exercise and so the agent needs to periodically check if the dog is beginning to tire.
Figure 2 shows a hierarchical task decomposition of the walk-the-dog task as described above. Ovals represent tasks and subtasks, while lines represent the “is subtask of” relationship between the higher oval (the parent task) and the lower oval (a subtask). Each of the subtasks in figure 2a, is further decomposed into additional subtasks in figures 2b-f.

The overall task of walking the dog is broken into five tasks that follow the pre-planned route and give the dog some exercise. Each of these consists of a set of leaf tasks that can be implemented based on the agent capabilities from task specification 1. For example, keep-dog-moving-forward, keep-dog-out-of-street, keep-dog-close-by and reel-in-leash-near-flowers are all leash control skills that monitor the dog’s position and tighten or loosen the leash at various times. Keep-dog-out-of-street monitors the positions of the dog and the street and tugs on the leash if they become too close to one another. Keep-dog-nearby allows the dog a certain amount of leash depending upon the agent’s location along its route.

Walk-toward-corner, walk-toward-park and move-across-when-clear are all agent navigation skills. Watch-for-flowers, watch-for-cars-at-crosswalk and wait-for-dog-to-return are perceptual skills based on known characteristics of the agent’s vision system. Throw-ball-to-open-area and get-ball-when-dog-returns are ball handling skills that make use of the agent’s reloadable ball launching system. Is-dog-tired is a perceptual skill that determines if the dog is tired of playing fetch. Of course, most tasks make use of multiple capabilities, such as throw-ball-to-open-area, which uses the vision system to detect a clear place for the dog to run and the ball launching system to throw the ball. For this discussion, the implementation of these tasks and the reasons for this specific decomposition are unimportant. What is important is that the leaf nodes all represent primitive actions based on the agent’s capabilities and the actions overall effect produces the right qualitative behavior.
Figure 2. Walk-the-dog Decomposition Hierarchy
Figure 3. Walk-the-dog Subtask Flow Diagrams
Figures 3a-f show the “flow of control” during task execution, i.e. the order in which sub-tasks are executed to achieve parent task goals. Subtask ovals that are stacked above one another indicate that these subtasks are executed in parallel. A solid arrow from one subtask to another indicates that when the left subtask completes, control transfers to the right sub-task. An arrow from a higher oval to a lower one indicates that the parent task executes one or more of its child subtasks. A dashed arrow from a subtask back to a parent task indicates that the child has achieved its goal and therefore the control flows back to the parent task. Figure 3a shows the basic flow of the walk-the-dog task as it has been described (go to the corner, cross the street, go to the park and play fetch with the dog). Figures 3c and 3e show that a number of subtasks are executed in parallel when the agent walks the dog to some location. Three subtasks control the dog to keep it safe (out of the street and nearby) and keep it headed forward. The final subtask moves the agent toward its destination. The route to the corner is lined with flower beds and so the agent must keep the dog out of them while walking to the corner. Figure 3b shows the 2 parallel subtasks that monitor for flowers and reel in the leash whenever they are detected. Figure 3b has no control flow back to the parent indicating that the subtasks will continue to execute until some event (external to this task) causes the agent to stop executing the parent task (and thereby stop executing its sub-tasks). In this case, the agent will continue to run the subtasks of keep-dog-out-of-flowers until walk-dog-to-corner completes. Figure 3d shows how the agent crosses the street. First the agent checks for oncoming cars, then if there are none it crosses the crosswalk keeping the dog close by. The play-fetch-with-dog task of figure 3f shows a series of tasks that play fetch with the dog. The agent throws the ball, waits for the dog to return with the ball and checks if the dog is tired. While the last subtask tests the dog’s status, there is no obvious
change in control flow based on this information. However, the tiredness of the dog is passed up to the play-fetch-with-dog task and that task determines whether to follow the dashed arrow of figure 3f (to throw-ball-to-open-area) or the dashed arrow of figure 3a (to walk-the-dog).

### 3.3.2 Decomposition Rationale

The purpose of this step in the methodology is to have the designer think about the relationship between the agent’s task and the agent’s capabilities. A hierarchical task decomposition can provide a form of information hiding [60] to some tasks. The tasks at any one layer of the hierarchy do not need to understand the internal workings of the tasks at the other layers, only the mechanism to communicate with them (see Section 3.7). The designer needs to create this hierarchy from both the top-down and the bottom-up. That is, the designer must think about what services a task at a certain layer of the decomposition should have available from the layer below. At the same time, the designer must consider how the goal-achieving behaviors of multiple tasks, at a certain layer, can be combined to present an abstraction at a layer above. The tasks at the lowest layer of the decomposition (referred to as “behaviors”) will come most directly from the agent capabilities, while the tasks at the highest layers will rely on the abstractions of tasks from all the layers below.

The creation of the task decomposition begins by designing the agent’s skills. What exactly constitutes a skill will depend on the agent’s given capabilities and the abstraction they present, i.e. how much they are a “black box”. For example:

<table>
<thead>
<tr>
<th>Task</th>
<th>Task Specification 2. Pour-a-cup-of-coffee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pour coffee from a pot in a coffee machine into a cup</td>
<td></td>
</tr>
<tr>
<td>Pour a cup of coffee</td>
<td></td>
</tr>
<tr>
<td>Coffee machine with coffee pot in machine, coffee cup</td>
<td></td>
</tr>
<tr>
<td>Coffee machine with coffee pot in machine, coffee cup</td>
<td></td>
</tr>
<tr>
<td>Pair of 6-DOF arms with attached grippers, visual sensor to detect pot/cup alignment</td>
<td></td>
</tr>
<tr>
<td>Pair of 6-DOF arms with attached grippers, visual sensor to detect pot/cup alignment</td>
<td></td>
</tr>
</tbody>
</table>
ple, an agent with an arm and gripper effector has the capability to \textit{grasp} objects, i.e. close the gripper “fingers” around an object, but pre-made software libraries might allow the same agent to \textit{pick up} objects (involving moving the arm to the object, positioning the gripper, grasping and then lifting the object). From the agent’s point-of-view, this “pick up” behavior is a black box. As a further example, consider the agent capabilities described in task specification 2. Given the agent hardware description, the designer might create behaviors to grasp objects like a cup or coffee pot, pull such a pot from a coffee machine, align a pot with a cup, and tilt a pot to pour coffee.

Once the agent’s basic behaviors have been designed, the designer must decide both how skills can be combined to achieve various task goals (bottom-up) and how various task goals can be decomposed into goals that can be achieved by other tasks (top-down). This can be done by considering the effects of executing various tasks (starting with the skills) in parallel or in sequence, as well as, how tasks can be broken down into sequences of subtask or groups of parallel-executing subtasks\textsuperscript{2}. For example, pouring a cup of coffee (see task specification 2) consists of the sequence of grasping the coffee pot’s handle, removing it from the machine, aligning it over the cup, and tilting it to the proper angle. Other tasks are composed of subtasks that execute in parallel, such that the desired behavior “emerges” [16]. For example, the driving task in task specification 3

\begin{table}[h]
\centering
\begin{tabular}{|l|}
\hline
\textbf{Task:} drive car to the store \\
\hline
\textbf{Environment:} car on road with other cars, lane markings, street signs, traffic lights, start in driveway \\
\hline
\textbf{Agent:} grasping/pulling effector, obstacle/landmark/lane detector, steering skill, color detector \\
\hline
\end{tabular}
\end{table}

\textbf{Task Specification 3. Drive-to-the-store}

\begin{itemize}
\item \underline{Task:} drive car to the store
\item \underline{Environment:} car on road with other cars, lane markings, street signs, traffic lights, start in driveway
\item \underline{Agent:} grasping/pulling effector, obstacle/landmark/lane detector, steering skill, color detector
\end{itemize}

\textsuperscript{2} The reader should note that the discussion of serial and parallel execution of tasks does not mean that the task decomposition is an and/or tree [64]. Task decompositions used in this methodology do not have “or” branches, though a particular subtask may select among various choices when deciding what to do.
might consist of steering to stay in your lane, avoiding other cars and watching for the next location where you need to turn. These activities must be done in parallel in order to drive safely and we consider them to all be part of normal driving. The designer should also look to create behaviors bottom-up, by executing groups of more primitive behaviors in parallel. For example, if the designer had watch-for-lane-changes and check-for-brake-lights behaviors, he might execute them in parallel to create the avoid-other-cars behavior. It is worth pointing out that all subtasks will ultimately decompose to some kind of hardware-triggered transitions in the agent’s electronics. However, such details are well below the “level of abstraction” that is important at design time and the decomposition should stop at the level of the agent capabilities.

The reader may ask, “since all agent behaviors ultimately arise from the execution of a collection of the agent’s skills, why not decompose the agent’s task into a ‘flat’ ordering of these skills?” While no argument can be made that a hierarchical decomposition is necessary for any task, as we will discuss later, it allows the designer to create an agent architecture that separates the agent’s deliberating and acting concerns to allow efficient and effective operation in dynamic domains.

For any given task, there are multiple feasible decompositions. Sometimes a particular decomposition will represent different possible orderings of steps in the agent’s task, or choices among alternatives available to the agent. For example, the pour-a-cup-of-coffee agent can align the cup with the pot or the pot with the cup. In this case, it probably doesn’t matter and so the designer can just pick one. However, a car driving agent may be able to take any of several routes and might want to select among them based on weather, traffic conditions, etc. In this case, the designer will want to think of the various decompositions
as being part of a larger tree, with a runtime decision as to which branch of the tree will have its subtasks executed. The designer can think of this larger tree as an and/or tree, where each node is a complete hierarchical decomposition of the type advocated in this methodology (that is, without decision points in the tree itself, but possibly within the nodes).

The designer should be aware that creating the task decomposition is an iterative process with the rest of the methodology. The answers to the questions in the remainder of the methodology are dependent on the task decomposition and so the designer can arrive at a situation where some other question cannot be satisfactorily resolved without changing the decomposition. For example, the designer may at first believe that a task will be used in two different parts of the decomposition. Later in the perception question (section 3.6), the designer discovers that the two situations in which the task is to be used, occur in suitably different parts of the environment so that the perception needs to be different in each case. This would involve changing the decomposition to have two different (situated) tasks instead of the original one.

Once the designer has a task decomposition, task flow diagrams should be created. This process should be assisted by the fact that when decomposing a task into subtasks, the designer is likely to have imagined an ordering of those subtasks.

For the most part, creating the decomposition and task flow diagrams is part of the art of agent design and this thesis has little specific to suggest. Software engineering provides a number of methodologies for decomposing tasks into smaller conceptual units, e.g. [11][23][67][84]. Some of the more popular methodologies are discussed in chapter 2.
3.3.3 Decomposition Summary

In this step the designer is to create a hierarchical decomposition of the agent’s overall task. This begins by deciding on the agent’s most primitive behaviors. Next the designer considers how the agent’s “top-level” goals can be broken into subgoals, and how the behaviors can be combined to achieve those subgoals. This step takes advantage of domain characteristics 2 and 4 (see sections 1.3.1.2 and 1.3.1.4) by exploiting the innate hierarchical structure of the task so that (in future steps) specialized representations for these tasks can be created.

A hierarchical decomposition is useful in two ways. First, the hierarchy gives shape to the final control structure of the agent (see Section 3.8). Second, tasks can hide operational details from their parent task [60] so that the parent task need not spend its time considering too many fine details.

3.4. Identify Task Roles

*What are the task roles?* After creating the task decomposition, the next step is to identify the *roles* in each task. Task roles are the aspects of the environment that effect the outcome of the task. Typically, they are the objects that the agent acts upon during the task (e.g. it’s “the hat” when the agent’s task is to “put on the hat”).

*What entities can fulfill those roles?* Once the task roles are determined, the designer must decide what entities in the agent’s environment can fulfill those roles.

3.4.1 Example Task Roles

Consider the environment shown in figure 1. The designer must first consider the entities in the environment that are important when walking the dog to the corner. First, the agent
needs to know about “the corner” in order to identify and navigate toward it. Obviously the agent also needs to have a dog. The agent also needs to be conscious of the flowers along the way and of the location of the street (or sidewalk edge) with respect to the dog. These are the primary entities (corner, dog, street and flowers) needed to get to the corner in accordance with the task specification.

Next the designer must decide what aspects of the environment are important when crossing the street. The agent needs to be conscious of oncoming cars, it needs to identify the crosswalk (to get across in the legal manner) and, of course, it needs the dog. After crossing the street, the task of walking the dog to the park is easier than walking the dog to the corner because there are no flowers along that part of the route. So, the agent need only be concerned with the location of the dog, the street and the park. Finally, when playing fetch with the dog, the agent must be able to detect open areas to throw the ball to, as well as detect the dog and the ball to retrieve the ball when the dog returns. Obviously, the dog is the important entity for determining if the dog is tired.

I have used various phrases such as, “know about”, “have the concept of” or “be concerned with” to designate the aspects of the environment that are important to the agent during a specific task. Describing a task with such phrases is a key to the designer that the associated aspect is a role in the task. That is, these phrases signify that some object in the environment possesses qualities that the agent must sense and respond to. There is a dog role in all the tasks because the agent must monitor the dog’s location and redirect the dog if necessary. If the agent has the perceptual capabilities to designate some portion of its environment as “the dog”, it can use that portion in actions with the task role “dog”. For convenience, I have repeated the decomposition of figure 2, with the roles in bold, in figure 4.
Figure 4. Walk-the-dog Task Roles
I caution the reader that while the hyphenated task name phrases of figures 2 and 3 appear to make the selection of roles obvious, the designer will not have these well-crafted names when developing the decomposition. One method that allows the designer to identify task roles and create English language task names is to describe the task’s goals in words. Often the nouns will be roles in the task. Skill in doing this will allow agent designers to create the sort of semantically meaningful task names used throughout this thesis.

However, even with a goal describing task name, all roles may not be obvious and chapters 4 - 6 contain many examples. Often, some roles arise from the interaction of tasks (which is partly an implementation issue). For example, the agent in chapter 4 has a task that detects obstacles and so this task has a role for an obstacle. However, another task the agent’s wheels also has a role for an obstacle because the task steers the agent around them. This second obstacle role was needed because detect-obstacles only finds obstacles, and expects some other task to steer around them.

The entities that can fulfill the task roles can also be determined from the environment (in the present example, the information about the environment is encapsulated in figure 1). First, there is the dog role. The task specification provides no details about our dog, except that the agent has a dog detector. So, the dog role will be filled by whatever entity in the environment is identified by this detector\(^3\). The corner must be that corner on the right side of street at the crosswalk. The park must be the park on the other side of the street. While walking the dog to the corner, the flowers can be either the tulips or the roses. The street will be anything beyond the edge of the sidewalk.

Note that from here on, I use boldface to denote the task role and plain text to denote ac-

---

3. This “dog detector” is given as an agent capability to simplify the discussion here, but it is an oversimplification of a complex problem.
tual entities in the world. So, **dog** represents *the* dog (not necessarily explicitly as discussed in section 1.2.2) that the agent is walking and while that dog may be just one of several dogs in the environment, its the important one.

The designer must choose the important portion of the street to fill the **street** role in the cross-street task. One solution is for the **street** to represent the area in between the two curbs (edges of the sidewalks) on either side of the street at the crosswalk. The **car** role can be fulfilled by any large moving object in the street. In this case, the role name is slightly misleading since the **car** does not have to be played by a car, but could be played by a truck, a bus, or a grand piano rolling down the street toward the agent and its dog. Anything that is large, in the street and moving toward the agent should be avoided.

The ball used to play fetch with the dog can be whichever ball the agent carries with it to the park, but the agent should make sure the dog returns with the same ball that was thrown (and this is done by the get-ball-when-dog-returns task). This means that ball will play the role **ball** in the ball throwing and reloading tasks. The open-area in the park can be any space which is clear of objects that the dog should avoid (more on how this is determined in Section 3.6.1). This open-area will then play the **open-area** role.

### 3.4.2 Task Roles Rationale

Roles are task specific descriptions of environmental aspects that influence the agent’s behavior, i.e. what the agent must sense and act upon. In a dynamic environment the agent cannot afford to be concerned with anything else [8]. This is why task roles are so important. They force the designer to pick out what is important in a particular task and allow the agent to concentrate its resources on just those entities [35]. This is not just a matter of efficiency, but of effectiveness. It is generally impossible for the agent to build a complete
model of the world, which it keeps up-to-date in a reasonably dynamic environment\(^4\) [49]. There is simply too much information available to the agent for the agent to extract it all and then decide what to use. The agent must concern itself with only the environmental aspects relevant to its current task. This allows the agent to be designed to accomplish each of its tasks using tight perception/action loops that involve determining information about object(s) associated with task role(s) and choosing corresponding effector commands.

The pour-a-cup-of-coffee task of task specification 2 requires some object to play the role of the coffee pot and some object to play the role of the coffee cup. Other environmental aspects are not important. Cups other than the cup that is getting filled are not important and neither are any of the myriad of other data which may be available to the agent’s sensors (e.g. the contents of the kitchen cabinets or the color of the floor).

At this point it should be made clear that a role’s “description of environmental aspects” does not necessarily refer to perceptual qualities, like red, or round, but possibly to qualities that only have meaning within the current task. For example, a navigation task might have to steer around obstacles while heading for a goal. Since obstacles and the goal determine the agent’s route, there should be \textit{obstacle} and \textit{goal} roles. Any entity fulfills the \textit{obstacle} role when it is positioned between the agent and its goal, regardless of its perceptual properties. For another task that same entity may play a different role or none at all. Of course,

\(^4\) I define a “reasonably” dynamic environment in a somewhat circular manner. That is, it is an environment where changes occur at a rate that precludes the agent from acting quickly enough to accomplish its goals if it tries to build a complete world model. This definition is clearer when viewed from the perspective of how slowly such an environment would have to change for modern computers to keep up. As an example, consider the robot Spot discussed in chapter 5. Spot uses 100\% of a 200MHz Pentium MMX to monitor an area of approximately 50 pixels square in a 256x256 pixel image. Spot can compute stereo and motion information about this area approximately 10 times a second. This allows tracking of the object in this space at a slow walk. If Spot were to perform his computations on the entire image, the process would be approximately 25 times slower. As computers get faster, this factor will be reduced, but the agent will always be able to handle a more dynamic environment by ignoring unimportant aspects of it.
perceptual information can be important as well. If the agent wanted some entity to play the role of sandpaper, it might select any object with the perceptual quality of a rough surface. So, the description of a task role may involve both perceptual information and task dependent information.

The designer must understand that roles are more than just descriptions of properties that entities in the environment must have, they contain semantic information. This semantic information allows tasks to be flexible because it tells the task how it should act with respect to certain entities without providing specific descriptions of those entities. The obstacle role above tells the designer something about the associated entity beyond a perceptual description, i.e. it is in the agent’s path and should be maneuvered around.

The question of what are the roles in a task is deeply entwined with the question of the entities that can fulfill those roles. I use the term “bound” to denote a role that has been associated with an object in the environment. That role is said to be bound to that object and that object is said to fulfill that role. Sometimes the task will only achieve the “correct” result if its role(s) is bound to a specific entity (drive-my-car is only done correctly if my-car is associated with the car owned by the agent). Other times, there can be multiple entities that can fulfill a task role. For example, the flowers role in the walk-the-dog example could be played by tulips or roses. For the pour-a-cup-of-coffee task, the coffee cup could be a mug, a tea cup or a styrofoam cup. Task roles need not even be associated with single objects. Typically, these objects must bear some spatial relationship to each other. For example, a task’s bowl-of-apples role might be fulfilled by a bowl object with at least one apple inside of it. Roles may also represent more nebulous entities, such as “the lawn” in mow-the-lawn.
It is also not necessary for the designer to exhaustively specify every entity that could fulfill a task role, rather a description of a class of entities can be used. For example, the **car** role in the watch-for-cars-at-crosswalk subtask of walk-the-dog can be bound to any entity fitting the “large and moving toward the agent” description. In the actual implementation of an agent’s perception system, a given perceptual description will always allow “recognition” of any one of a set of entities. This is because these routines will necessarily contain descriptions of perceptual qualities to search for and not unique object identifiers. The number of entities that match a given description will depend on the capabilities of the agent, the implementation of the agent’s perceptual system and the amount of noise in the perceptual process.

The designer must also take note of when one role depends on another. This means that any information the agent gets about the entity associated with one role effects the information the agent has about the entities associated with other dependent roles. This often occurs between a role in a parent task and roles in one or more of its child subtasks. For example, the **street** in cross-street depends on the **crosswalk** in cross-crosswalk-when-clear and watch-for-cars-at-crosswalk because the important part of the street is defined by the curbs on either side of the crosswalk. So position information about the curbs (and hence the crosswalk) affects the agent’s notion of the position of the street. This primarily occurs when one task uses more abstract properties of some entity than the other tasks, yet each task has a role that should be associated with that entity (or some portion of that entity). For example, assume that the drive-to-the-store task had a **car** role for the car to be driven and the pre-drive inspection contained a check-tire subtask with a **tire** role. There would be a dependence between this role and the **car** role because if any tire is flat, the car is un-
drivable. So, the agent may store the fact that the car is “undrivable” in the *car* role without any indication of why. The fact that this tire was flat would be stored in the *tire* role. Note that a roles can be dependent on more than one other role, such as if the check-tire task had separate roles for each of the car’s tires. The *car* would then be dependent on all 4 *tire* roles. Also, the dependency need not be symmetric, e.g. the “undrivability” of the car depends on the state of the tires, but the state of the tires is not determined just because the car is undrivable (the car could be out of gas). Identifying dependencies is important in representation design because the representation must be structured to communicate information between dependent roles. I return to this issue in section 3.7.

Since roles are very context dependent (i.e. their meaning is derived from the agent’s current “situated” activity - as defined in section 1.3.2), tasks can be made simple and reusable. If the agent needs to walk to a number of destinations, it can keep executing the walk-to-destination task and just change the entity associate with the *destination*. By using the available task context, designers can create tasks that need not perform inferencing about the environment (see Section 3.8 for a discussion of inferencing by tasks and when it must be avoided). Rather, because the semantic meaning of an object associated with a role is pre-defined, the task’s actions can be pre-defined also. The semantics of roles allow the designer to make situational control rules that are flexible. The roles can be bound to different entities, but the task will treat them the same.

In general, there is a trade-off between the flexibility of a task to bind its roles to many different entities and the “situatedness” of that task. The more restrictive a task is about the specific objects to which it will bind its roles, the more likely it is that the designer can optimize a task because it only has to operate on those entities. However, the more entities
that a task can work with, i.e. bind to its roles, the easier it is to reuse that task in different situations. This is particularly important for agents with planners because the tasks that the agent can execute will determine the “operators” that the planner considers. The fewer there are, the easier it is to plan, and of course, fewer tasks need to be implemented.

When deciding on task roles, it may be easiest for the designer to start with the task roles for the leaf nodes in the decomposition hierarchy. As we will see later, the leaf subtasks tend to be executed by the most responsive layer of the agent’s architecture, which places strict time constraints on the amount of processing that can be done. The computationally straightforward nature of the leaf subtasks means that the task roles are usually easy to identify. These subtasks can often be aptly described by simple verb-noun phrases like grasp the coffee mug or open the door.

3.4.3 Summary

The designer must identify task roles because the agent must focus its resources on only the aspects of the environment that are important to the current task. Designing agents this way addresses domain characteristic 1 (see section 1.3.1.1) because it allows the designer to concentrate the agent’s processing when the dynamic environment makes it impossible to monitor everything. Identification of the objects in the environment that can fulfill the roles is important because information about those objects will structure the remaining steps in the methodology.

3.5. Representation of Task Roles

What role bindings are shared between tasks? Recall that a task role/entity binding refers to the agent having selected an entity in the environment to play a specific role in its current
task. Thereafter, the task’s actions will take place on that object. For example, the pour-a-cup-of-coffee task has a **cup** role. The agent selects a particular cup to use (from the set of all cups that fit the task role’s perceptual description), it is considered bound to the **cup** task role and that entity will be filled with coffee. Sharing refers to roles in different tasks being associated with the same object. It also refers to dependent roles that hold different information or a different abstraction of the same entity. The designer must also remember that tasks across levels of the decomposition hierarchy may share roles.

**What information about the entity bound to a task role is needed for the task?**

**For what roles would it be useful to develop an explicit representation?** Explicitly representing a task role means that some data structure exists to hold information (from the previous question) about the entity bound to that task role. See section 1.2.2 for a discussion of implicit vs. explicit representation.

**How often should the task role information be verified?** That is, how often should the agent check that its stored information is still valid? Throughout the remainder of the dissertation, I use the term “maintenance” to refer to the active process of verifying the validity of a piece of information (by allocating perceptual and computational resources to the task of determining that information). Maintenance of information is in contrast to “storage” of information, which refers to passively holding some data in memory. I sometimes refer to deciding how often a task role’s information should be verified as deciding on the rate of maintenance for that role.

### 3.5.1 Example Task Representation

In determining what role bindings are shared between tasks, we need to examine the task environment and our decomposition. The **dog** role occurs in almost all the tasks of walk-
the-dog, not surprisingly, because the dog is central to the agent’s purpose. All these tasks should have their dog role bound to the same dog and so that role binding is shared. By the route shown in figure 1, we can see that the agent needs to pay attention to only one of the corner, crosswalk or park roles at a time, in order to accomplish the walk-dog-to-corner, cross-street and walk-dog-to-park tasks, respectively. So these roles are not shared between those tasks. However, these roles are shared within the various “subtrees” of their particular task’s decomposition (see figure 2) and so we must consider those subtrees in more detail.

Walk-dog-to-corner consists of controlling the path of the dog with its leash while navigating to the landmark named corner. At the same time, the agent must keep a lookout for flowers and possibly do additional leash manipulation to keep the dog away from them. The corner in walk-dog-to-corner should be the same physical corner as in walk-toward-corner, so corner is shared between the parent and child tasks. Besides dog and corner, the important roles in the tasks that are active at this time are the flowers and the street. The flowers are shared between the subtask that creates the role binding (watch-for-flowers) and the leash control skill (reel-in-leash-near-flowers) because the leash control skill reacts when flowers is bound by pulling the dog away. Obviously, both subtasks need to refer to the same flowers. The street in keep-dog-out-of-street will be bound to different parts of the street as the agent walks along. It should correspond to the portion of the street within range of the dog, given the current leash length. These roles (flowers and street) are not shared with any other subtasks because other subtasks do not try to control the dog with respect to the flower beds (they keeping the dog moving forward, etc.)

5. The reader may ask why keep-dog-out-of-street isn’t broken into watch-for-street and reel-in-leash-near-street, in the same manner as keep-dog-out-of-flowers? It is an arbitrary decision I am making here mainly for purposes of discussion.
The **street** in cross-street is different than the **street** in keep-dog-out-of-street. Cross-street’s **street** is bound to the crosswalk at the corner, i.e. the area between the two curbs. This **crosswalk** is shared between the watch-for-cars-at-crosswalk and cross-crosswalk-when-clear tasks because the agent should cross the same crosswalk that it has just determined to be free of **cars**.

Walk-dog-to-park is similar to walk-dog-to-corner in that the **park** role is shared between the parent and the walk-toward-park subtask. The **street** is, again, not shared with the other subtasks because the other subtasks are concerned with different aspects of the dog’s motion.

The final task in walk-the-dog is play-fetch-with-dog. The **dog** is again a shared role between this task and its subtasks, as is the **ball**. The agent must throw the ball and retrieve it from the dog (and it ought to be the same ball in both cases).

Throughout this chapter I have mentioned that the position of entities bound to task roles is the information that the agent’s tasks need. This is because the agent’s tasks involve moving itself and the dog along a certain path and not getting too close to particular points (e.g. flowers). Therefore, position is the information that should be computed for task roles.

Now the designer must consider which of these shared roles should have some explicit form of representation. Since the **dog** role is shared among most of the tasks and subtasks in this domain, I argue that it should be represented. The dog’s position is important for all the top level tasks. Since the dog will be moving all around the agent while the agent is monitoring entities bound to other task roles, such as the **street** and the **flowers**, having a dog representation will allow the agent to remember the dog’s position even when the dog cannot be seen [14]. Information indicating the dog’s position must be shared by all the
subtasks of walk-dog-to-corner and walk-dog-to-park. Providing an explicit representation of the dog’s position not only allows for sharing among parallel subtasks, but for passing of dog information between the subtasks.

In addition to the dog role, the corner role and the park role in the walk-dog-to-corner and walk-dog-to-park tasks need to have some form of representation. The position of the entities fulfilling these two task roles are used to drive the agent’s locomotion system. Again, since the agent will at times be monitoring the dog, the street or the flowers, it will not always be directly perceiving the corner or the park. The “limited field-of-view” argument [14] for representation applies here and is discussed in section 3.5.2. Basically, the agent cannot perceive both the street and the flowers at the same time because they are on opposite sides of the sidewalk and the agent’s vision system only covers a limited area. However, the agent cannot forget about the existence of one of those entities, just because it is looking at the other.

The case for representation of the flowers and the street in the keep-dog-out-of-flowers and keep-dog-out-of-street tasks is similar, since the positions of these entities are used in leash effector control. Detecting these entities and tracking their positions facilitates the computation of geometric properties (such as proximity) between the entities and the dog. While this is consistent with psychological literature, e.g. [19][63][76], the most compelling reason to represent these entities is the limited field-of-view argument [14].

The street in the cross-street task may not need representation. As stated in section 3.4.1, the street is actually determined by the positions of the curbs on either side of the crosswalk area. For the watch-for-cars-at-crosswalk task, the agent needs to be able to see the entire crosswalk (i.e. both curbs, the area in between and some of the street before the crosswalk)
at the same time in order to make sure no cars are coming. Since all important entities are in view for the duration of the task, they need not be represented explicitly. However, for the cross-crosswalk-when-clear task, it may be useful to have a representation for the curb on the other side of the street (the side where the park is) because the agent can use this to servo on. Since this task runs in parallel with keep-dog-close-by, the agent may have to divert its attention to the dog and no longer be able to perceive the curb (again, the limited field-of-view argument). So, the agent need not represent **street** because no information about the street is needed by later tasks. All that needs to be represented to get the agent across the street is the crosswalk, which will be associated with the curb on the opposite side from the agent.

It may seem that the **ball** used in play-fetch-with-dog also does not need representation, as long as the dog actually fetches it each time. This is because throw-ball-to-open-area assumes the ball is in the agent’s ball throwing effector and get-ball-when-dog-returns always puts it there. Since the dog presumably carries the ball in its mouth, knowing the dog’s position is sufficient to find the ball during get-ball-when-dog-returns. If the dog fetches the ball, the agent need not be concerned with where the ball lands after it is thrown and so none of the play-fetch-with-dog subtasks need to store any information about the ball. However, if the agent does not begin the walk-the-dog task with the ball in its launching effector, but say in a carrying pouch, a ball representation might be useful to store whether the ball was in one of the known fixed locations (pouch or launcher) or “thrown”. The agent need not represent the **open-area** because the dog will go there and get the ball. Once the ball is thrown, the agent does not need to know any more information about that area (and in fact, a new open-area will be selected on each loop of the fetch task, so information stored about
any particular open-area will not be useful for long).

Now the designer must consider the rate of representation verification. Since the agent uses the positions of entities for effector control, the agent needs to verify the positions at a rate as close as possible to the rate of effector control. This rate is needed so that changes in important environmental entities will be detected and responded to as quickly as practical. Consider the dog’s position, which as mentioned above, is the information that must be known about the entity associated with the role called **dog**. The agent executing the keep-dog-out-of-street task might, for example, pull on the leash with strength proportional to the distance between the entities associated with the **dog** and the **street** representations until those entities are far enough apart.

Keep-dog-moving-forward might involve monitoring the dog’s position and pulling on the leash until the dog is in front of the agent. The effectiveness of the agent’s leash control depends on effective monitoring of the dog’s position. Since the positions of the entities associated with **corner** and **park** are also used as feedback in effector control loops, this information should also be verified at the same rate that these loops issue effector commands. Verification of the ball’s position can be done much less often since the position is either one of two fixed locations, or “thrown” and the transitions between these states take place at well-known times (since they occur only by the agent’s actions).

If the agent has the computational and perceptual resources to verify the information in all representations at the rate of effector control, then the verification design is done. However, most agents have only a limited perceptual field and so, regardless of computational resources, cannot verify information stored in all representations because they cannot perceive all relevant entities. Now the designer must make trade-offs between timeliness of in-
formation and the cost of shifting perceptual resources to different areas of the environment.

In the walk-dog-to-corner subtask there are several parallel activities. Recall from figure 1 that as the agent is walking toward the corner, it must keep the dog out of the street on one side and the flowers on the other. The agent has to switch between verifying the position of the entities associated with **corner**, **dog**, **street** and **flowers**. Since the position of the corner, street and flowers only change (with respect to the agent) due to the agent’s own motion, the agent can estimate their locations with its proprioceptive sensors. A reasonable algorithm might be to update the position of the flowers when the dog is on the right side of the agent and the street when the dog is on the left side. Obviously the dog should be monitored in both cases.

Since the bindings of both the **street** and **flowers** will change as the agent moves, it is not necessary to monitor the associated entity’s position when the dog is on the opposite side of the sidewalk (since whatever entity is currently bound to the role will likely be unimportant the next time the dog crosses over there). The agent must also periodically check its position relative to the corner. This can be done opportunistically, when the dog walks in front of the agent (when both the dog and corner are in the field of view) or based on the accuracy of the proprioceptive sensors (when the dead-reckoning is known to become inaccurate).

For the cross-street task, the agent can begin by trying to bind **car** to any large object moving toward the crosswalk. After a suitable length of time, if **car** has not been bound, the agent can stop trying to bind it and begin moving across the street. If the agent has a representation for the curb on the other side of the street from the corner (the other side of
the crosswalk) then there is a trade-off between monitoring the dog and watching the servo point used to cross the street. If not, the agent might just move forward one street width and watch the dog.

Walk-dog-to-park has the same trade-off between verifying information about the dog and the park as between the dog and the corner in walk-dog-to-corner. Again, the agent can update the park’s position opportunistically or when dead-reckoning becomes inaccurate. During play-fetch-with-dog, the agent does not need to monitor the position of the ball since the dog will retrieve it. In fact, the agent may not even monitor the dog since it is going to come back with the ball. In this case, the agent just needs to perform a search in its local area to determine when the dog has returned.

3.5.2 Representation Rationale

Task roles are variables that become associated with (bound to) entities in the world, much like variables are bound to values in a program. For some roles it is useful to “save” this association rather than recompute it whenever it is needed. Saving the association refers to having an internal representation, i.e. a data structure, that stores information about the entity bound to the task role. Representation can be a powerful tool for an agent designer because it can be used to share data between tasks and provide perceptual continuity over both time and distance [13]. We examine the sharing of roles between tasks, since when multiple tasks have roles that should be associated with the same object, it makes sense for them to share a single role (and its binding) instead of each maintaining their own binding.

Role sharing is a principle reason for having an explicit representation for a role. I will now discuss the advantages and disadvantages of representation in detail.

The creation and maintenance of role/entity bindings is a large part of the computational
load of the sensor/effect control component of an agent’s architecture (recall the discussion of PA systems in section 1.3.2). The designer can reduce this load by taking advantage of multiple tasks that use the same binding (so each subtask need not go through the binding process). Often parallel subtasks in an emergent behavior scheme [16] will each effect the same entity. The pour-a-cup-of-coffee task requires that the pot be aligned over the cup and tilted at the correct angle. The tilting and aligning tasks occur in parallel and each refers to the same pot.

Task role/entity bindings are also shared between sequential tasks. The pour-a-cup-of-coffee agent must grasp the coffee pot’s handle, remove the pot from the machine and then do the alignment/pouring task. The pot role should be bound to the same object in each of these for the overall task to make sense. After the first task makes a binding, that binding should be used by the subsequent tasks.

One caution to the reader is that this thesis uses consistent names for all roles that are shared between tasks. In other words, when two tasks share a role the role has the same name in each task. However, the designer must often do some abstraction to see when multiple tasks have roles that can be filled by the same entity (and hence those tasks can share the binding). Imagine that the skills for the pour-a-cup-of-coffee agent are being designed by different teams. If the skill that removes the vessel with the coffee from the coffee machine represents that coffee container with a role named pot, while the skill that pours from the vessel into the cup represents the coffee container with a role named kettle, the designer must make sure that pot and kettle are associated with the same object. As another exam-

---

6. Note that even the implicit roles will need to be taken into account in the coding of the agent’s various behaviors. That is, the implicitly represented roles will still be used by the agent to decide on its actions. A task will calculate some information about the entity fulfilling an implicit role, it just won’t store it.
ple, imagine that the pour-a-cup-of-coffee agent designers are borrowing a skill from a previously created agent that poured a glass of juice. This juice pouring agent had a skill called pour-from-pitcher-to-glass that had **pitcher** and **glass** roles. The pour-a-cup-of-coffee agent designers might use this skill by sharing the binding of the **pot** role in remove-pot-from-machine with the **pitcher** role in pour-from-pitcher-to-glass\(^7\).

The decision of what information about a particular entity is important for the task obviously depends on the needs of the task. In the walk-the-dog example, positions of entities are important because all the agent’s actions are based on the proximity of various objects (e.g. How close is the dog to the street? How close is the dog to me? Is there a car coming at me?). In fact, I argue that position is the fundamental piece of data that needs to be stored (and kept up-to-date) for every represented entity in the domains of interest to this thesis. This is because, as stated in section 1.3.1.1, agents designed by this methodology are meant to act in dynamic domains where current sensor data is more relevant than previous sensor data. The tasks addressed by this thesis involve robots navigating around and reacting to changes in their environment. In order to determine anything about the environment, an agent first needs to know where to look, i.e. the position to direct perceptual resources, and then other properties can be determined. This is not to say that some tasks may not need other data about entities. For example, in the drive-to-the-store task of task specification 3, when the agent is stopped at a traffic light, it must monitor the color of the signal. The agent’s brake effector control process may monitor the color of the object associated with the **traffic-signal** role and release the brakes when it turns green.

Once the designer has decided what information about an object associated with a role is

\(^7\) This is analogous to software reuse [74].
needed by the task, role representation must be considered. Role representation has several benefits. First, representing a task role allows the agent to deal with a limited field-of-view [13]. The limited field-of-view argument for representation says that important entities that can be outside the agent’s sensory field during a task should be represented so that the agent doesn’t just forget about them. For example, the walk-the-dog agent’s vision system has a limited field of view and so cannot perceive the roses and the street simultaneously. By representing the position of the flowers and the street, no matter which one the agent is looking at, it can react (based on the stored data) if the dog suddenly bolts for the other.

Note that not all information that a task needs to know about an entity must be stored in that entity’s representation. The traffic-signal representation used by the drive-to-store agent might just store the signal’s position so the agent can find it again if the agent looks away while waiting at the red light (to change the radio station perhaps). The signal color may not be stored since once it turns green, the agent releases the brake and is no longer concerned with the signal.

Another advantage of representation occurs when a role is shared. Role sharing comes in two different forms. First, when multiple tasks act with respect to the same entity, representation allows those tasks to share the role data structure (and hence the cost of creation and maintenance). Another form occurs between the dependent roles identified in section 3.4. This form of role sharing refers to sharing information about the entity bound to a role because it effects the information the agent has about entities bound to other roles. Representation can be useful here because the tasks that manipulate the data structures of dependent roles can reflect any new information in these roles representation with a minimum of communication with other tasks. If the roles are not represented, each time a task T wants
information about an entity bound to a dependent role, it must gather data from the other
tasks with roles on which task T’s role depends. Depending on the architecture, this com-
munication can be expensive. Some roles represent more abstract properties of the environ-
ment than one or more other roles on which they depend. For example, the bowl-of-apples
role may store the fact that the entity associated with the apple role is inside the entity asso-
ciated with the bowl role, while the apple and bowl roles themselves just store the posi-
tions of those entities. These dependent roles should be represented because these more
abstract properties are often expensive to compute and so the agent would like to be able to
store the result for some time.

The designer should not think that representing all task roles is beneficial. There are costs
associated with representing and not representing. These issues are at the heart of the ques-
tions of what information about the bound entity should be represented and how often that
information should be verified. Proponents of the reactive approach argue that the world is
its own best model [15] and so nothing needs to be stored internally. Information can al-
ways be determined by consulting the world when necessary. Implicit in this argument is
that the relevant entities can always be kept within sensor range, and if this is true for a par-
ticular task, the designer may consider not representing that task’s roles. However, always
computing a piece of information from the current sensor data requires the continued allo-
cation of perceptual and computational resources. The more information stored in a repre-
sentation, the more information the agent has access to when the associated entity is not in
viewable, but the more expensive the maintenance. The agent must strike a balance be-
tween the amount of information stored and the rate at which it is verified since either can
overburden the agent’s finite computational power. In much the same way that deciding on
task roles caused the designer to focus the agent’s tasks on only what was currently important, the selection of what entity information should be stored and how often it should be verified serves to focus the tasks even more.

The designer should note that sometimes the agent’s capabilities obviate the need for representation of a role. For example, if the dog walked in the walk-the-dog task had a GPS transmitter in its collar and the agent had a GPS receiver, the dog might not need representation because it would effectively always be in sensor range.

Deciding how often information needs to be verified is also task dependent. There is a time/usefulness trade-off between a high rate of maintenance (and thus high usefulness of data) and computational load. Recall that the “rate of maintenance” refers to how often a piece of information is verified. So, the higher the rate of maintenance on a piece of data, the more useful that data is, but the more expensive it is in terms of perceptual and computational resources. Note that this maintenance need not be periodic. For some information, it may be sufficient to verify it whenever the agent can schedule that computation in with its other jobs.

Traditional reactive systems [17] place emphasis on usefulness of information and so pay the maximum computational price. However, this methodology advocates examining the agent’s task and capabilities to determine acceptable rates of maintenance. In fact, there exists the possibility that certain pieces of data cannot be verified given the current sensory view. This is where the trade-off becomes a more complex decision. If all entities associated with represented task roles are always within the agent’s perceptual field, then the rate of verification presents only a computational cost. However, if an entity can move out of the perceptual field (either through its own motion or the agent’s) the agent must direct its
perception system to a different location in order to verify information about that entity. This means that other important entities may no longer be in view and so their information cannot be maintained at any computational cost.

Maintenance can be arbitrarily complicated, possibly involving planning. Going downtown to verify that the grocery store is still there is an extreme example. So, the designer must create a “maintenance scheme”, i.e. a procedure for keeping representation verified at required rates, that balances task needs for up to date information, computational cost, and limited sensory horizons (areas perceivable by a sensor at any one time).

Two other factors that are important in representation maintenance are “opportunism” and “effector vs. non-effector control”. “Opportunism” refers to the agent happening to get the entities associated with multiple task roles within its perceptual field simultaneously. The agent should be designed to take advantage of this, such as when the dog walks in front of the dog-walking agent and the agent can see both the corner and dog in the same view. Both representations can be updated at the same time.

“Effector vs. non-effector control” refers to whether or not a piece of information stored in a role is used by that role’s task to directly control effectors or for some other purpose. The reel-in-leash-near-flowers task uses the position of the dog and the flowers to directly compute commands for the leash effector (how hard to pull and in what direction). Other pieces of information that the agent may compute about the entities associated with task roles may not be used by effector control tasks. Deciding if the pot has enough coffee will determine if the pour-a-cup-of-coffee agent should even proceed with the pouring action, but does not directly influence the motion of an agent effector. This sort of non-effector control data will typically need to be verified much less often than the effector control data.
and so the agent designer should bias the maintenance process so that most of the agent’s time is spent verifying the effector control data.

Since the majority of the agent’s maintenance computation will be devoted to effector control data, the role representations used by the leaf tasks should store no non-effector control information. For these tasks, non-effector control data is only excess baggage whose maintenance will slow down the task’s operation. In order to understand this, consider that all information stored in a role representation will need to be verified at some point. If it doesn’t, then the information is unaffected by the dynamic nature of the environment and there is no benefit to explicitly representing it. Instead, the agent designer should design that knowledge into the agent’s tasks with an implicit role. So, since all role data needs to be verified, and the rate of verification determines the usefulness [14] of the role information to its task, effector control tasks should not expend effort maintaining non-effector control information.

It is perhaps not obvious that representation should be deleted when a task completes, providing the represented role is not shared with another task. Once a representation is no longer useful, the agent should not maintain it, and so deleting these representations reduces the agent’s maintenance work. Even when a role is shared between tasks, the designer may want to have the agent delete that role’s representation if the sharing tasks are not executed closely in time. If the work required to recreate and re-bind the representation is less than the work to maintain the representation in between tasks, it makes sense to delete the representation.

3.5.3 Representation Summary

The designer must decide which task roles should be explicitly represented in the agent’s
control program. The two main reasons to have an explicit representation are to provide information about important entities outside the agent’s current perceptual field and to share data. These address domain characteristics 3 and 5 (see sections 1.3.1.3 and 1.3.1.5) because the former provides a symbolic model that other tasks can use to help control the agent and the latter provides a means to deal with large-scale space [43].

I have argued that a piece of information that must be stored in every role representation is the position of the associated entity, if for no other reason than to direct the agent’s perception system to the entity to determine other information. This takes advantage of domain characteristic 6 because the agent can (at least coarsely) estimate positions. The designer must also decide on an appropriate representation maintenance scheme. There must be a balance between the usefulness of the stored information about each currently relevant entities and the amount of computation involved in establishing that information.

3.6. Perception

*What information can the agent extract from the environment to recognize the entities that should be bound to the current task’s roles?* The agent’s perceptual capabilities will form the baseline for what it can identify in the environment. Those capabilities and any task dependent information must be encoded into perceptual descriptions of task roles and associated recognition processes. The term “primitive recognizable”, or PR, is used to denote any entity from the set of entities that the agent will match directly to a given perceptual description. “Directly” means that object recognition is based on only the agent’s given perceptual capabilities and not on any combinations of, or relationships between, separately recognized entities.
How does the duration of the various role/entity bindings effect the perception system?

What level-of-detail (or resolution) is required in the information of the representation?

Resolution refers to the precision of the data stored in the representation.

3.6.1 Example Task Perception

For simplicity of exposition, I have specified that the dog walking agent has the perceptual capabilities to directly identify most of the entities that can fulfill its task roles (see task specification 1). The dog, the ball, the corner, the park, flowers, curbs and cars can all be directly recognized by the agent’s perception system and so are PRs. The street, however, must be found in relationship to curbs. From figure 1, we can see that the keep-dog-out-of-street task can be accomplished by keeping the dog away from the closest curb, which can be identified by the agent’s sidewalk edge detector. Also, flowers come in two types, roses and tulips (see figure 1) and so the agent must be able to recognize either of these. The watch-for-flowers task will involve looking for roses, then looking for tulips, then back to roses, etc. The perception system will need to handle multiple perceptual descriptions for the same role.

It is tempting to say the open-area in throw-ball-to-open-area can be recognized by the lack of any of the entities that the agent wishes to keep the dog away from, i.e. no flowers, curbs or cars. However, such “negative” recognition processes are problematic because even if the agent finds no object that conflicts with the open-area description, there may be other entities present. The agent does not want to throw the ball into an open pit just because there are no flowers, curbs or cars there. These negative processes are similar to PROLOG’s negation as failure model [68] because when the agent can not determine that any known objects is in an area, the area is determined to be “open”. These kinds of prob-
lems pervade perception systems on autonomous agents and I do not seek to eliminate them here. If an agent’s perceptual capabilities do not allow it to identify necessary entities in the environment by another means, then the agent’s capabilities are deficient and the best the designer can do is be aware of the issue.

Next, the designer must examine the duration of the role/entity bindings for the roles involved. Clearly the dog role has the longest binding duration (the length of the task). The corner and park remain bound for the length of the walk-dog-to-corner and walk-dog-to-park subtasks. However, the street and flowers bindings change as the agent moves down the street. The crosswalk in the subtasks of cross-street remains bound for the duration of those tasks as does the ball in the subtasks of play-fetch-with-dog.

The designer would like to take advantage of the fact that the dog binding lasts for so long by “foveating” the dog’s position to speed up the perceptual computation that determines its position. “Foveating” refers to processing only a limited portion of the input image around the entity’s (last known) position. Since the dog should always correspond to the same dog, the agent need not look over a large area for other potential dogs, once the initial binding is made. The same is true of corner and park. However, the fact that the environment is dynamic and all these entities cannot be kept in view simultaneously may cause this strategy to be modified slightly. When the agent looks away from the position of an entity for some time, it may have to examine a larger (or entire) portion of the perceptual stream when it returns to view that entity because the estimate of the entity’s position may have become incorrect and an expanded search may be necessary.

The street and flowers roles cannot be foveated since those roles can potentially be re-bound to other flowers or parts of the street as the agent and dog move. However, since the
crosswalk in the cross-street task remains bound to the same entity (the curb) for the entirety of the task, it can be foveated and tracked. The ball does not need to be foveated because the agent does not have to continually track its position (since its position is stored as “in pouch”, “in launcher” or “thrown”).

Finally, the designer must address the level-of-detail needed in the representation’s information. For most of the task roles, the agent needs to know the position of the associated entity. When that role is a landmark (corner, park, or crosswalk) the accuracy of this position value can be less than the accuracy of the position stored in roles used for leash control (dog, street, flowers). This is because when starting toward any landmark, only the general distance and direction are useful. The agent might dead-reckon the position of a landmark, meaning the accuracy of the position estimate will decrease. Since coarse metrics are effective in this domain (as mentioned in Section 1.3.1.6), provided the agent periodically verifies its estimates with its vision system, it can still navigate to the landmarks.

However, since the dog moves of its own accord and the street and flowers can be bound to different entities at any time, estimating their positions with proprioceptive sensors is risky. Of course, the agent will have to do just that if, say, the dog is not in the same view as the corner, but the corner’s location needs to be verified (such as when the agent believes its estimate of the corner’s position can no longer be trusted based on its understanding of the limits of its dead-reckoning capabilities). Lastly, as mentioned above, the ball’s position is stored as one of three states, which is a fairly course level-of-detail. This level is acceptable because of the agent’s simple ball launching system.

3.6.2 Perception Rationale

This step seeks to impart some structure to the agent’s perception system based on the
agent’s task and capabilities. Ultimately, the agent’s perception system is charged with the creation and maintenance of the bindings between task roles and entities in the environment. The computational demands of maintaining task role information in a dynamic environment means that the designer must carefully structure the perception system to be efficient. The first step in this process is to consider how the task specification and agent capabilities determine what information the agent can (and should) extract from the perceptual stream to recognize entities to be bound to task roles. In some ways, this is both a top-down and a bottom-up process like creating the task decomposition.

There are some roles for which perception must come top-down from descriptions of particular objects that can fulfill them and the agent must somehow use its capabilities to detect this description. In other words, the walk-the-dog agent cannot associate the dog role with the mailbox just because it can easily detect the mailbox. On the other hand, perception for roles such as car in watch-for-cars-at-crosswalk can be thought of in a bottom-up manner. Any capability the agent has can be combined to find “cars”. If the agent has a motion detector, then its reasonable to use it to identify cars, and a “bumper detector” would work if there were no parked (non-dangerous) cars. Some iteration with the task decomposition and role selection steps may be necessary if the entities that can possible fulfill a role are not detectable by the agent. Ultimately, if the agent cannot identify objects that it must act on for its tasks, then the agent’s capabilities are insufficient.

As another example of how task and capabilities determine what perceptual data should be used in the recognition process for a role, consider the whack-a-mole agent of task specification 4. This agent has the ability to detect and localize motion as well as determine color. So, the agent could identify a mole as a brown object protruding above the game surface.
Alternatively, the mole could be any fast moving object within the game area. Since the task specifies that the game contains only moles (all of which should be whacked), the later perceptual description could suffice, but not if the motion computation cannot proceed at the rate that the moles pop up and down. Note that in either case, each mole is a PR.

The designer may need several recognition processes to detect the entity that should be bound to a role, for a variety of reasons. Some task roles are not filled by single objects, such as the **bowl-of-apples** role that is fulfilled by a bowl and some apples with the correct spatial relationship between them. The designer will have to have thought about combinations and relationships among the objects that fulfill roles when identifying dependent roles in section 3.4. For example, consider the agent of task specification 5 whose task is to pick up a covered pot from the stove. The agent may have a specialized “pot with lid” detector that can identify the vessel and cover as a single object, or PR. On the other hand, if the agent can identify the pot and lid as separate PRs, the **pot-with-lid** might be identified as a **pot** with a **lid** above it. If the agent could determine the position of the pot’s handles, then for the pick up task the agent could identify the pot by the location of its handles (since handle position is the necessary information for that task’s effector control).

There are other times when multiple perceptual routines are needed because the perception of all objects, and particularly these compound objects (“objects” consisting of individually identified entities) is situation dependent. Consider a lamp and a lightbulb. The

![Task Specification 4. Whack-a-mole](image)
agent may be able to identify each individually, but when the lightbulb is screwed into the lamp’s socket, the “socketed” relationship cannot be perceived except from an overhead or underneath perspective. Similarly, a penny on top of the Empire State building would appear very small when perceived from the ground, if it could be seen at all.

Another important issue is that objects can change their perceptual characteristics when assembled. A knife and a wooden knife-set holder have one appearance when separate, but when the knife is put in the holder, only the knife’s handle is visible. A sugar cube dropped into a cup of water loses its cube shape. Of course, these two categories of perceptual difficulties are not so different. Both problems are the result of the fact that objects look different in different states. A “different state” may result from the object being at a different location (the penny), in a different spatial configuration (the light bulb), partially occluded (the knife), or being physically altered (the sugar cube). The effect of all this is that the agent may need multiple perceptual descriptions of an object to recognize that object in different situations.

Once the designer has decided how entities associated with task roles can be recognized, two necessary optimizations must be made by considering the effects of role/entity binding duration and required precision of stored information. These “optimizations” are vital to operate effectively in a dynamic environment.

We examine the duration of the role/entity binding because if the agent needs to monitor some entity that is bound to a task role for a long period of time, it may be advantageous to foveate the entity to detect changes in entity properties needed for tasks. For many agents,
e.g. [4][8], this means tracking the entity’s position. Reducing the area of the perceptual field that must be searched, decreases the time needed to determine the needed entity information. This increases the potential update rate for the entity’s representation, which for tracking means the entity can move faster relative to the agent (and the agent will not loose it). However, if a role is bound to several different entities during the duration of the task, foveation may be less useful as the agent will still need to examine a (larger) portion of the perceptual field for other entities that could be bound to the task role. For example, suppose the play-wack-a-mole task consists of a single subtask, hit-whackable-mole. The **whackable-mole** role will be rebound to a variety of different moles during the game. Limiting the **whackable-mole** search process to a single mole hole will only allow the agent to hit one of the moles (and thus get a low score). There may be other cases where the agent both foveates the currently bound entity and examines a larger portion of the perceptual field for other entities that should be bound to the task role. For instance, the agent may implement hit-whackable-mole by foveating the current **whackable-mole** and monitoring whether it is still out of its hole. As long as the mole is “up”, the agent should continue the hit action against this particular mole. At the same time, the agent is processing the current visual field for moles that can be bound to the **whackable-mole** role when the current mole has been hit or retreats into its hole.

Determining if a binding duration is “long” depends on the execution time of various tasks. Long binding durations arise in two different situations. One is when a single task with a long execution time binds a role that stays bound to the same entity for a large portion of that task’s duration. The second way a role can have a long binding duration is for several shorter, sequential tasks to share a role that they all expect to be bound to the same
entity. Of course, “a long execution time” and “a large portion” are still ill defined, but basically if a role is bound and the agent then does not need to look for other, more appropriate entities to bind the role to, foveation of the target can save processing time.

The last question in this section addresses the fact that some tasks need more precise information about the entities bound to roles than other tasks. Some information can be determined from multiple sensors. For example, the position of an object can be determined by vision or by sonar. Some sensors provide more coarse data and so there is less data to analyze. Other sensors are more precise and still others have a larger perceptual field. All sensors can only detect certain environmental properties. Since sonar only provides positional information, data from a typical robot sonar system is much coarser than visual data. However, it typically covers a larger perceptual field. Proprioceptive sensors provide only a small amount of information, but the size of their perceptual field is not a concern. The designer needs to examine precision requirements to create a representation maintenance scheme. That is, a designer must determine when different sensors can be used to determine information and develop some strategy for allocating sensor resources. In the walk-the-dog example, the agent used a combination of proprioceptive sensors and vision to maintain its representation. This was done because the agent cannot use its visual resource to acquire all task relevant entities all the time and the lower accuracy sustained while an entity was out of view was manageable.

Answers to this question, along with the designer’s decision on the required rate of maintenance (see section 3.5) define four basic classes of role maintenance requirements.
Figure 5 shows the space of role maintenance requirements divided into four quadrants based on precision and maintenance frequency. Exactly how a designer decides what constitutes “high” and “low” precision or maintenance frequency depends on the agent’s task and will be relative to the needs of the various tasks the agent must perform. However, placing the needs of a particular role into one of these classes will help the designer create an appropriate representation maintenance scheme for the agent. Additionally, each quadrant can be further divided into the same four classes and the design decisions discussed below will apply to those finer divisions as well.

The four quadrants are as follows. Quadrant 1 contains roles that whose associated information needs to be verified frequently and whose information must be precise. These roles put the most constraints on the agent’s maintenance scheme and they require the largest amount of time for the high accuracy sensors. For example, the dog in walk-the-dog needs to have the camera trained on it as often as possible because its position needs to be verified frequently and the camera is the agent’s most precise position-determining sensor. Quadrant 2 represents roles that need to be verified often, but do not require much precision in

---

8. The information must always be accurate. That is, the agent will perform worse with inaccurate information than with correct information. The issue is, how exacting does that information have to be? When position is specified, in how small a region does the agent have to localize the entity.
the information they contain. Hopefully, the agent can just use another, less precise sensor to handle these roles as the most precise sensors will be tied up with maintaining the roles in quadrant 1. For example, the agent in chapter 6 avoids obstacles with sonar. Sonar provides coarse position estimates for obstacles (compared to the visual estimate of the position of the agent’s destination provided by the agent’s camera), but the avoidance task does not need very precise information anyway because it tries to circumvent obstacles and not squeeze through tight spaces. Roles in quadrant 3 are those that need to be updated infrequently, but need to contain precise information. Often these roles are ones that depend on other roles and hold an abstraction of the information the dependent(s) contain. For example, the pot role in the pour-a-cup-of-coffee task needs to store the pot’s position in one task and the amount of coffee in the pot in another. The amount of coffee must be precise, to tell if another cup can be poured, but it need not be checked often (only before each new cup). Roles in this category will vie for time from the precise sensors with the roles in quadrant 1. The designer must create a scheme where by quadrant 3 role data can be occasionally computed, with the bulk of the time being spent determining quadrant 1 role data. Finally, quadrant 4 contains roles that neither need to be updated frequently, nor need to contain precise information. Although there may be tasks that have roles fitting into this category, the designer is advised to consider whether or not such roles really need (explicit) representation. Roles in this category may contain information that is needed by a scheduling or planning system and so while their information may need to be determined, it may not need to be held in a representation in the action-oriented portion of the agent’s architecture. For example, a role that stores the answer to the question “is it raining?” might be in this quadrant if the agent believes that rain showers do not end quickly (and thus this data will not
need frequent updates since it will be true or false for some time) and if the agent is not concerned with how hard it is raining (so precision is not important). Instead of being stored in a representation, this type of information could be sent to a planning system and stored there, but this is beyond the scope of this work.

There is also a computational dimension to representation maintenance because computing arbitrary information can take arbitrary amounts of time. If the agent wants to verify some complex property of an entity at a high rate, with high precision, it must have the computational power to do so and the designer can do little to help if it does not. This brings up the question of effector vs. non-effector control information from section 3.5.2. The computational difficulty of determining some role information can effect all categories of roles. Roles in quadrants 1 and 2 would be effected by any computation that slowed down the rate at which the agent’s resources were available to verify their data. Roles in quadrant 3 often store information that is expensive to compute (which is why they are updated infrequently) and the agent will need to schedule that computation so as not to interfere with other maintenance. Even roles in quadrant 4 can be problematic if their maintenance computation needs to be done in an environment where the loss of cycles to other role updates causes the agent to miss events. “Anytime algorithms” for determining the required properties of the environment allow the designer to make trade-offs between precision and computation time. The amount of time available can be allocated between the roles so that their information is at least of the desired precision relative to the other roles. Of course, scheduling of anytime computation is NP-complete [87] and this issue is not addressed by this thesis. In addition, compositions of system components with different anytime algorithms does not guarantee that the system as a whole behaves like an anytime algorithm [87].
3.6.3 Perception Summary

In general, the designer will want roles in the leaf tasks of the task decomposition to be fulfilled by PRs because these can more easily be maintained at rates commensurate with the effector control loops that implement these tasks. This may cause the designer to iterate between deciding which entities can fulfill a task role and which entities are PRs.

In this step, the designer must decide how the agent’s perceptual capabilities can be used to recognize entities that can fulfill various task roles. For the represented roles of the leaf tasks of the decomposition hierarchy, it is important that the entities bound to the roles be PRs so that their representations can be maintained effectively. The duration of role/entity bindings can allow the perception system to “foveate” the area around a bound entity and thereby decrease processing time. Also, the level-of-detail required for particular data allows the agent to allocate its resources to where they are most needed. The designer may have to go back to a previous step of the methodology (perhaps to the task decomposition or the selection of task roles) and try a redesign if all the current representations cannot be maintained at the required precisions/rates. All these considerations are meant to address domain characteristic 1 (see section 1.3.1.1) since that characteristic places demands on the agent’s computational power and thus its ability to react.

3.7. Communication

What information is important in inter-task relationships? In other words, what data, computed by one task, is needed by another? In Section 3.5, roles that are shared between tasks were identified and so role information is clearly be exchanged between such tasks. At this point it is important to also think about data other than the information stored in role
representation (from section 3.5) that tasks may need to exchange. The results of (the actions of) previous tasks or the progress of other tasks in the system are examples.

3.7.1 Example Task Communication

In the walk-the-dog task, there are several kinds of information that are communicated between tasks in this domain. First, there is the information about the environment that the roles store (much of which has been mentioned in various descriptions of the tasks). Tasks obviously communicate position information for the entities associated with their roles. This communication takes place between parallel executing tasks that all base their actions on that information (such as between the subtasks of walk-dog-to-corner and walk-dog-to-park) and between sequential tasks such as depicted in figure 3a.

When the first subtask of figure 3a (walk-dog-to-corner) is executing, it maintains the position of stored in **corner**. When the task is complete, this information is communicated to cross-street to help establish the initial position of the crosswalk. Cross-crosswalk-when-clear reports the final position of the crosswalk (stored in **crosswalk**) to cross-street upon completion so that this information can move on to walk-dog-to-park. There it is used to determine the agent’s trajectory to the park. **Dog** is maintained by many tasks over time and when each completes executing, that role information provides the next task with the position to continue monitoring the dog.

Tasks also communicate both perceptual descriptions and binding information for the entities to be bound (or that have been bound) to their roles. In this domain, many roles can be bound to only one entity and so a perceptual descriptions of the entity could be built into the tasks. However, perceptual information is still communicated. Consider the **flowers** role. Watch-for-flowers will bind the **flowers** role when flowers are in sight, and a percep-
tual description of those flowers will be communicated to reel-in-leash-near-flowers. This is useful because, as shown in figure 1, the tulips are farther back from the road than the roses and so the dog can be given more leash near them. The watch-for-cars-at-crosswalk task needs to communicate the fact that the car role has not been bound to an object (after trying for some time). This triggers cross-crosswalk-when-clear. As another example, suppose the agent were programmed with descriptions of several dogs that could be walked. The walk-the-dog task would pass a description of the current dog to all its subtasks, which would in turn pass the description to the leaf tasks. In one of these, the binding between the dog and dog will be created and subsequent tasks can use the perceptual description to maintain the dog’s position (although the agent may examine a smaller portion of its sensory stream).

In addition to passing information about the entities bound to task roles between tasks, the tasks exchange some notion of the confidence they have in that information. For example, if the keep-dog-out-of-street task has been observing the dog, the confidence in the position stored in that representation will be high. This may allow the watch-for-flowers or walk-toward-corner tasks to redirect the agent’s vision system to try and verify the flowers or corner roles. Confidence is a function of the time since the agent has acquired the role’s entity with its vision system and provides a means for implementing the representation update algorithm discussed in section 3.5.1.

Tasks also communicate some data to indicate a task’s action’s result. Most of the agent’s tasks just need to report success or failure and the next task in the sequence needs to decide what to do with “failure”. I have not specified what happens if any task “fails”, except for the is-dog-tired task, for which failure of its test means it should continue playing fetch.
However, imagine an agent with more sophisticated abilities to deal with failure. Now action results can be used to determine which task to execute next. For example, the agent could walk the dog to the corner, but the dog could break free and run away. The agent should not continue across the street, it should try and get the dog back. Note that the position of the corner alone was not sufficient to determine the success or failure of the walk-dog-to-corner task. The agent reached the corner, but it no longer had the dog.

Progress reports are another communicated item that represents a task’s estimate of how well it is proceeding toward its goal(s). This form of communication often occurs between tasks executing in parallel. For example, avoiding the mailbox on route to the corner (see figure 1) is accomplished by the walk-toward-corner task. When the agent is in the process of avoiding this obstacle, the task notifies keep-dog-out-of-street that the street should not be in view and watch-for-flowers that the flowers should be or vice versa (depending on the direction of avoidance). Both walk-dog-to-corner and walk-dog-to-park should tell keep-dog-moving-forward when they are avoiding obstacles so that the dog is not forced into an obstacle.

3.7.2 Communication Rationale

The purpose of this step is to help the designer structure the task role representation. The information communicated between tasks should be part of the role representation because it is either about the entities bound to the role or about actions taken on those entities. There is an intimate relationship between actions and entities because actions take place on objects in the world, not arbitrary points in space. Of course humans are capable of taking actions seemingly without any relation to objects in the world (e.g. grasp at the air), but these are hardly the sort of actions that allow robots to do useful work. Actions do not make sense
if they do not effect the physical world and roles are action-specific descriptions of the physical world (this stance is softened somewhat in succeeding chapters for situations where the agent’s sensors cannot detect the needed entity). So, by examining the types of entity information that tasks exchange to support their actions, the designer can determine what data needs to be in the role representation ⁹.

The designer can determine a structure for the role by examining what is communicated between tasks. Obviously, the information about the entity associated with the role (from the questions of section 3.5) will be communicated because it is what tasks need to know to accomplish their action(s). In addition, there are four principle classes of role information that tasks use and may need to communicate. They are *perceptual descriptions, actions, confidence* and *progress*. These classes of role information can be summarized as follows.

*Perceptual descriptions* are routines used by associated recognition processes to examine the sensory stream for entities that possess the “described” characteristics. These entities can fulfill the role. Such descriptions are often exchanged between parent and child tasks. Tasks, particularly leaf tasks, are often made “reusable” by allowing them to successfully operate with a number of different entities bound to a role. The pour-a-cup-of-coffee agent needs not have separate skills to manipulate mugs and tea cups, if it can operate with the cup role bound to either. The parent task typically has the information to decide what entity in the environment should be acted on by the child task and can therefore create an appro-

---

⁹. Admittedly, there is an engineering issue here as well. Since most of the information that is communicated between tasks is role information, it is efficient to add information to the role representation that is perhaps not strictly relevant to the role, but rather to the actions being taken on the role. For example, action requests, results and progress are information about the internal state of some process executing an action, rather than information about some entity in the world. However, since roles only exist to support the execution of these acting processes, it is not such a leap to say that the roles should carry all information that need be communicated between tasks.
appropriate perceptual description and communicate it to the child. The child task will either bind the role (if it is a leaf task) or create a new description appropriate for its child task. Consider the pick-up-the-covered-pot agent of task specification 5. The pick-up-covered-pot task might pass a perceptual description of the **pot** to the first of its subtasks (grasp-the-pot) and that task may break down the pot description into PR descriptions, such as **handles**, which are passed to the grasp-pot-handles leaf task.

Since there is necessarily a set of entities that match a perceptual description, a child task may report information about the actual bound entity back to the parent. For example, the pour-a-cup-of-coffee agent may report whether the **cup** role was bound to mug or a tea cup, so that the parent task can determine if there is enough coffee in the pot for that type of vessel. In the walk-the-dog agent, reel-in-leash-near-flowers and watch-for-flowers communicate the actual binding to the **flowers** role (to the tulips or roses) to determine the amount of leash the dog can use.

**Actions** are role information about the action to be performed and the results of that action. A role’s representation may or may not contain the “action to be performed” portion of the action information, depending on the implementation of a particular agent. For example, an agent may possess a skill that is capable of performing multiple actions on the roles that are passed to it. An agent with a grasping system might be able to hold a cup by the handle (to drink from it), by the cup itself (to wash it) or by the cup’s base (to invert it into the dishwasher). Agents without such multi-purpose skills may not need such action information in the role representation. However, most skills require some form of parameterization based on the agent’s current state (see chapters 4, 5 and 6 for many examples). Action parameterization can be complex, especially when coordinating multiple effectors.
For example, an agent with two arms that has to shovel snow may require separate parameters for actions to be executed by each arm effector so that snow could be thrown to the left or to the right.

For any action, the role representation should also contain the results of the action just performed so that the next task can determine the outcome of the previous task. While it may be possible for a task to determine the results of previous actions by examining the state of the world (perhaps by examining the entity data stored in roles), the results of an action are dependent on the context in which it is executed and that may not be adequately captured by the role representation. For example, an agent that tried to open a door and failed because the door was locked may want to report this fact to other tasks because by observing the environment, those tasks can only determine that the action failed (since the door is still closed), but not why. It may be helpful to have an action result stored in the role representation even when the information can be determined by inspecting the environment because it prevents other tasks from having to execute elaborate perceptual routines to infer information that is known to the acting task. In addition, as we’ll discuss in Section 3.8, doing a lot of inference can hurt the responsiveness of certain tasks.

Confidence denotes the agent’s belief that the information stored in the role’s representation is still useful [14]. Confidence is valuable for action because it helps determine how to allocate perceptual resources (to verify information in which there is low confidence) and select courses of action (to operate on the entity in which there is the most confidence). Confidence can be defined in a number of ways. When the role’s associated entity is out of the agent’s field of view, the designer may decrease confidence because the entity can move on its own and even if it did not, proprioceptive sensors are typically not as accurate
as direct perception. Confidence can also be expressed as a measure of how well an entity matches a role’s perceptual description. For example, the drive-to-the-store agent may have a role for a particular street where it has to make a left turn. As the agent approaches the street, it may be too far away to make out the street sign. So, the agent might have only 50% confidence that the upcoming street sign is the correct street. Confidence is not just whether a role is bound (though it can be) because in this case, the agent had bound the street-sign to a role, but it may end up not being the sign the agent really wanted. In chapters 4 - 6, we will see several examples of how confidence can be implemented for agents with different capabilities.

Finally, progress is a measure of how a task is proceeding. Progress can be used by other tasks to monitor the communicating task. Tasks may adjust their actions based on the progress of other tasks. For example, the drive-to-the-store agent may have one subtask that handles the navigation and another that watches for the store. If the navigation process has only have a general idea of how far it is to the store, but not the store’s actual position, it can report progress toward the store to the monitoring task. The monitoring task can then devote increasing time to looking for the store, as the agent draws closer.

It is, of course, possible for a designer to create tasks that communicate information other than that identified above as necessary for task inter-relationship. However, this information is extraneous and must be some “internal” state of the task. This data can only be about the task’s internal computations and not about important elements of the agent’s environment, otherwise it would be stored in the task’s role representations. Such “internal” data should not be communicated between tasks because it provides no semantic information about the environment. Significant inference or translation machinery would have to be
built to interpret the internal state data \textit{from each task}. Instead of this, I argue that groups of tasks that communicate should express any important information about the environment or their actions in their role representations so they can communicate through a common role structure.

When creating role structures for an agent’s various tasks, the designer must take into account the dependent roles identified in section 3.4. These roles contain information that depends on information stored in other roles. Role information must be communicated to tasks with dependent roles as part of the agent’s inter-task communication. This means that representation structures must contain information in a form that a task with dependent roles can interpret in order to correctly update its roles. Task communication and appropriate role structures are the means for establishing epistemological links between dependent representations.

3.7.3 Communication Summary

This section discussed what information is communicated between tasks. Since tasks operate on roles, they should be communicating information about the entities bound to those roles, specifically the represented roles (since unrepresented roles are not needed outside of their task). This leads to the proposal that representation should be the channel by which tasks communicate (many other agent architectures, e.g. [10][65][71], have also used representation for communication).

This section outlined a structure for role representation. Roles have six basic components that I will call “Index”, “Property”, “Identify”, “Action”, “Confidence” and “Progress”. These components are summarized in table 3.2.
At this point, it is worth emphasizing that Index and Identify are independent of each other. That is, a single entity can play several roles (so the Identify component would be the same, but the Index component would change) and a single role can be fulfilled by several entities (so the Index component would remain fixed while the Identify component changed). An example of the former would be if the pour-a-cup-of-coffee agent had to throw away the cup after it was used. During the pour-a-cup-of-coffee task, the cup would be associated with the role, **cup**. However, for the throw-trash-away task, the same cup might be associated with **trash**. An example of the latter occurs in the drive-to-the-store task when different signs get assigned to the **street-sign** role for the read-street-signs task.

### Table 3.2: Representation Component Summary

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>A task dependent label that indicates this representation’s role in the current task, i.e. its role name.</td>
</tr>
<tr>
<td>Property</td>
<td>Information about the bound entity that is stored in the representation (and hence is a small amount)(^a).</td>
</tr>
<tr>
<td>Identify</td>
<td>A perceptual description used to detect the entity associated with the role in the sensory stream. Identify may contain the perceptual data used to make the initial role/entity binding and information about how to maintain the binding after it has been made.</td>
</tr>
<tr>
<td>Action</td>
<td>The action that the agent wishes to take on (or with respect to) the entity bound to this role. It also contains the results of the action on the role (when the action completes) and additional parameters needed by the action (beyond those in the Property component).</td>
</tr>
<tr>
<td>Confidence</td>
<td>A measure of the agent’s belief that the information stored in the role representation is correct, i.e. it is a measure of usefulness [14].</td>
</tr>
<tr>
<td>Progress</td>
<td>Denotes how well the current, ongoing action on the entity associated with this role is proceeding.</td>
</tr>
</tbody>
</table>

a. For all the representations used by the walk-the-dog agent, this component could be called the “where” component because it stores the position of the bound entity. However, I use the term Property because different tasks (especially at different levels of the decomposition hierarchy) may store different information in their representations.
This role structure addresses domain characteristics 1, 2, 3, 5 and 6. The roles are designed to carry only the information used by the current task and thus be maintainable in a dynamic environment (1). The roles are used for communication between architectural components (2 and 5) and finally they hold quantitative data (6) about the environment beyond the agent’s sensor horizon (3).

### 3.8. Architecture

*How should the agent’s tasks be laid out in its architecture?* The designer must now create an architecture that uses and manipulates roles discussed in the previous sections.

#### 3.8.1 Example Task Architecture

Since this thesis targets multi-layered architectures (see section 1.3.2), the walk-the-dog tasks must be divided into layers. The tasks must be partitioned such that they can effectively use the representations designed thus far. This means that tasks the control effectors and bind their roles to PRs should be placed in the most responsive layer, i.e. the PA layer. This includes all the leaf tasks of figures 2b - 2f. These tasks should be in the PA layer because the PA layer’s tight perception/action loops can keep representations bound to PRs up-to-date.

The other tasks may or may not be in the PA layer. The more tasks that the PA layer needs to handle, the more likely it is to be overburdened and thus reduce the agent’s reaction time. These tasks are better off in a second layer of the architecture where they can activate the PA layer tasks as needed.

#### 3.8.2 Architecture Rationale

The previous steps of the methodology have guided the designer in choosing what to rep-
resent and the trade-offs associated with representation. The purpose of this thesis is to
guide the design of representation, not architecture. However, agents need architectures
and so this section discusses the impact that representation may have on the structure of an
agent’s architecture. An issue that is fundamental to both representation and architecture
design is that of “cycle time”. “Cycle time” is the time for a task to perceive the environ-
ment (including representation maintenance), select an action and execute it. Executing the
action does not necessarily mean running it to completion, but rather issuing effector com-
mands that move toward the task’s goal within the current environment. The next time
through the perception/action loop, the task will choose new effector commands and these
may continue the previous “action” or they may start a new one.

Representation should be designed so that it can be maintained at a rate that is effective
for the particular task using the representation. The impact that this has on architecture is
that a task must be executed by an architectural component that has a rapid enough cycle
time to maintain that task’s representation at rates required by the tasks. This thesis assumes
that the designer will be using a layered architecture and these architectures map nicely to
the “role hierarchies” that tend to arise when dependent roles represent portions of the en-
vironment at different abstractions. This means that if a role is dependent on one or more
roles in one layer, that role’s task should be in a higher layer. The abstract property stored
in that role will typically be expensive to compute and so would slow down the cycle time
of the lower layer. Tasks that require high rates of representation maintenance should be
executed by the PA layer, but this must be tempered by the computational requirement of
that maintenance. Tasks that can get by with slower maintenance should probably be exe-
cuted by other layers of the architecture because the more complex the task that the PA lay-
er has to be able to execute, the more likely the layer will be to slow down and ruin the responsiveness of other tasks.

The division of tasks into layers, will also depend on a number of other engineering issues, such as latency/bandwidth of inter-task communication, capabilities of layers of the target architecture, amount of task inference and computational power required. Communication is perhaps the issue most effected by representation. Tasks that execute in parallel and share a role (i.e. communicate role information) should be in the same layer of the architecture because intra-layer communication is generally at a higher bandwidth and a lower latency. Tasks at different levels of the hierarchy communicate information about the roles at different levels. For example, suppose role1 depends on role2 and role3, which are in tasks at different layers of the architecture. There will necessarily be communication as part of the maintenance of role1 and this communication represents the “link” between these roles. The designer will want to consider the frequency of the communication when placing the tasks in the architecture this way. Natrajan [57] discusses methods of automatically updating role1 when either role2 or role3 is updated and such a scheme involves even more communication. The designer needs to consider how often role information is exchanged under their particular maintenance scheme. Often, roles in tasks at different levels of the decomposition can be at different layers of the architecture because the lower layers of the architecture maintain the important details needed for effector control, while the higher layers maintain other data that needs to be updated less frequently.

3.8.3 Architecture Summary

In this step, the agent designer must try and structure the tasks in the decomposition hierarchy into “layers” of the agent architecture. The first step was to divide the agent’s tasks
into PA layer and non-PA layer tasks. The PA layer tasks would be executed by a component of the agent architecture that is responsive enough to events in a dynamic environment to appropriately maintain those tasks representation. The non-PA layer tasks may themselves be divided into layers based on a number of characteristics. Often the final structure of the non-PA portion of the architecture will be determined by implementation concerns (e.g., latency/bandwidth of inter-task communication and computational power required).

This methodology is concerned with the design of representation systems, not the design of agent architecture. Since each agent needs an architecture, this step approaches the question of architecture design from the perspective of representation’s effect on it. However, the final structure of the architecture is mostly an engineering issue, not a contribution of this work.

3.9. Post Methodology

After completing the methodology questions, the designer will still have work to do. At this point, the designer should have a layered system based on the agent’s capabilities and the task decomposition. The designer should know what roles exist in the agent’s tasks and what entities in the agent’s environment can fulfill those roles. Knowing which role/entity bindings are shared and how long these bindings last will help the designer decide what roles need representation in the agent’s architecture. Answering questions about the information that the agent must store and maintain in these representations will help structure the agent’s perceptual system. Understanding what information must be shared or communicated between tasks will help determine the flow of information within the system. Finally, based on the flow of information and the nature of the tasks, the designer will partition them into sets which can be executed by different “layers” of the agent’s architecture.
At this point, the designer must implement the agent architecture such that it can effectively manipulate the system of representation that has been designed. The agent must use its perception abilities to maintain the information stored in the role representations and the agent architecture must have a means of exchanging task role binding information between tasks (at the same or at different layers).

3.10. Methodology Summary

This section repeats the important aspects of the methodology. The methodology is meant to answer three questions about agent representation (sections that most directly address those issues are given in parenthesis, though answers to the questions in those sections may depend on the answers to questions in previous sections).

What should be represented (see section 3.4)? How should that representation be structured (see section 3.7)? How should that representation be maintained (see sections 3.5 and 3.6)?

The design can answer all these questions by answering the questions of the methodology, which are repeated below.

- What are the agent’s primitive skills?
- Which tasks can be decomposed into sequential subtasks? Which can be decomposed into parallel subtasks?
- What are the task roles?
- What entities can fulfill those roles?
- What role bindings are shared between tasks?
- What information about the entity bound to a task role is needed for the task?
- For what roles would it be useful to develop an explicit representation?
- How often should the task role information be verified?
- What information can the agent extract from the environment to recognize the entities that should be bound to the current task’s roles?
• How does the duration of the various role/entity bindings effect the perception system?
• What level-of-detail (or resolution) is required in the information of the representation?
• What information is important in inter-task relationships?
• How should the agent’s tasks be laid out in its architecture?

The final role structure developed by the methodology is also repeated in table 3.3 (see section 3.7 for the culmination of the arguments for it).

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>A task dependent label that indicates this representation’s role in the current task, i.e. its role name.</td>
</tr>
<tr>
<td>Property</td>
<td>Information about the bound entity that is stored in the representation (and hence is a small amount)(^a).</td>
</tr>
<tr>
<td>Identify</td>
<td>A perceptual description used to detect the entity associated with the role in the sensory stream. Identify may contain the perceptual data used to make the initial role/entity binding and information about how to maintain the binding after it has been made.</td>
</tr>
<tr>
<td>Action</td>
<td>The action that the agent wishes to take on (or with respect to) the entity bound to this role. It also contains the results of the action on the role (when the action completes) and additional parameters needed by the action (beyond those in the Property component).</td>
</tr>
<tr>
<td>Confidence</td>
<td>A measure of the agent’s belief that the information stored in the role representation is correct, i.e. it is a measure of usefulness [14].</td>
</tr>
<tr>
<td>Progress</td>
<td>Denotes how well the current, ongoing action on the entity associated with this role is proceeding.</td>
</tr>
</tbody>
</table>

\(^a\) For all the representations used by the walk-the-dog agent, this component could be called the “where” component because it stores the position of the bound entity. However, I use the term Property because different tasks (especially at different levels of the decomposition hierarchy) may store different information in their representations.

This role structure addresses the domain characteristics laid out in section 1.3.1 because the roles are:

• designed to carry only the information used by the current task and thus be maintainable in a dynamic environment (section 1.3.1.1)
• specialized to the needs of particular tasks (section 1.3.1.4)

• used for communication within and between architectural components (sections 1.3.1.2 and 1.3.1.5)

• designed to hold quantitative data (section 1.3.1.6) about the environment beyond the agent’s sensor horizon (section 1.3.1.3).

This chapter presented a design methodology for autonomous agent with representation systems that specialized to the tasks and capabilities of the agent. The methodology is a series of questions to guide the designer in creating representation systems that are efficient and effective for use in dynamic domains.