NORTHWESTERN UNIVERSITY

The Intelligent Classroom: Competent Assistance in the Physical World

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

David Franklin

EVANSTON, ILLINOIS

June 2001

Abstract

How can we make robotic systems that physically interact and cooperate with people in interesting ways? Such systems will be useful to people because they understand the tasks they are to be used for and can figure out to help out in them. I refer to such systems (both robotic and purely electronic) as *competent assistants*: systems that are able to aid a user in his work because they understand what that work is.

This dissertation describes a competent assistant that advances the state of the art for physical interaction: the Intelligent Classroom, a prototype automated lecture facility that serves as its own audio/visual assistant. The Classroom domain provides an interesting level of interaction and also exhibits many of the interesting challenges of mobile autonomous robotics research. Key contributions of this research include:

- Techniques for connecting the low-level sensors and actuators to a plan-recognizing symbolic execution system.
- Demonstrating how a good level of intentional cooperative interaction can be achieved without explicitly reasoning about goals.
- A voice-operated slide-advancing system. Jabberwocky uses speech-recognition results to determine what slide a speaker wishes to discuss, allowing a speaker to dynamically change his presentation to suit the needs of his audience.

Acknowledgements

"Aren't you done yet?"—four words that are sure to tweak a raw nerve in any graduate student who is fighting his way towards a Ph.D. Certainly, if left to my own devices, I would still be hearing that question today, tomorrow, and right up until the day the computer science department finally kicked me out. Fortunately I was not on my own.

My first thanks go to my advisory duo of Kris Hammond and Jim Firby. Jim got me off to a good start in thinking about robotics and interaction at the University of Chicago—many of the best ideas in this dissertation sprang from our discussions. Kris adopted me when it was time to move on to Northwestern and has spurred me on to actually finishing. He has been an enthusiastic supporter of my work and has taught me the importance of being able to articulate my ideas in ways that anyone can understand.

Also, I thank the other members of my committee: Larry Birnbaum, who always finds an interesting new way of thinking about a problem and tells a great story, and Ian Horswill, who lets me think of myself as a roboticist and encourages me to keep my symbols grounded.

The key to my staying (mostly) sane throughout the whole graduate school process is certainly my group of friends in the InfoLab. Josh, my fellow Intelligent Classroom researcher, who shares my vision of an intelligent space and patiently puts up with my abuse. Robb, Shannon and Jay, who intuitively know when to encourage, when to argue, and when to just go out for a walk and grab a coffee. And all the other members of the lab, past and present: Andy, Louis, Xiaobin, Vidya, Robin and Marko.

Outside of Northwestern, I must thank Charmane and Larry, who inspired the "Psychological Warfare Brain Trust" that helped me fight through the last few barriers. Bruce and Karen, who provided a soothing combination of encouragement and music. Alain, who entered graduate school at the same time I did and will exit at the same time as well. Way to go, "Dr. Man!"

Finally, I thank my family and friends back home in Seattle. My Mom, who actually read (and commented on) my dissertation and who is always a great inspiration to me. My sisters, who keep me humble with the knowledge that, to them at least, I will always be a kid. Steve, who shares my belief that you can be both a computer geek and a hip jazz musician.

"Am I done yet?" Yes, I am.

Table of Contents

List of Figures

Chapter 1

Introduction

In the world of mobile autonomous robots, human/computer interaction has been almost exclusively limited to interactions of the form: (1) the human tells (through command, gesture or programming) the robot what to do, and (2) the robot tries to do the human's bidding while not running into anyone or anything. Even the earliest conceptions of robots were highly interactive. Rossum's Universal Robots (from the Karel Capek play "R. U. R." [15], which coined the term *robot*) aided people in everything from menial labor tasks to dealing with business correspondence. Even Robbie (the earliest and most primitive robot in Isaac Asimov's "I, Robot" [6]) was interactive to his core, serving as a child's nursemaid. Though incapable of speech, Robbie was very attentive to the needs of his charge, both physical and emotional.

Without suggesting that Science Fiction writers should dictate research agendas, I am convinced that they have an important point in this instance: interaction should be at the core of robotic systems. Such systems are useful to people because they understand the tasks they are to be used for and can figure how to help out in them. I refer to such systems as *competent assistants*, systems that are able to aid a user in his work because they understand what that work is.

Consider a master carpenter building furniture with an assistant. If the assistant is a complete novice in woodworking, the master carpenter will have to carefully describe every action that he wishes the assistant to take. In many cases, he will even need to physically demonstrate exactly what he wishes him to do. This method of interaction is both inefficient and potentially frustrating for the master carpenter. Although he knows exactly what he wants to do and how the assistant can be most helpful, he must spend a great deal of his time and effort in telling the assistant what to do and correcting him when the assistant fails to understand. Typical human/computer interactions are remarkably similar to this situation.

But what happens if we replace the novice assistant with a relatively experienced furniture builder? This assistant will be familiar with the ways that carpenters work together. He will know how high to hold a piece of wood so that the other carpenter can cut it. He will know where to put his hands so that they are out of the way and less liable to get hurt by an errant hammer swing. As a result, the master carpenter will be able to just go about the task of constructing the furniture, while the assistant helps out at the appropriate moments. Although there will be times when the master carpenter will need to explicitly instruct the assistant, these will be in exceptional (rather than typical) moments.

Many researchers are working on building competent assistants for software and internet tasks [43, 58, 69, 23], but task-based interactive work in the physical world has been much more limited [8, 54, 13, 37]. In this dissertation I discuss a competent assistant that advances the state of the art for physical interaction: the Intelligent Classroom, a prototype automated lecture facility that serves as its own audio/visual

assistant. Before detailing the Classroom's particular functionality, it seems logical to trace how my research focus moved from mobile autonomous robotics to the Intelligent Classroom. Many of the problems addressed by the robotics research community are also crucial to the Classroom. In implementing the Intelligent Classroom, existing robotic research was used as a starting point upon which to add cooperative interaction.

1.1 From Robots to Intelligent Classrooms

Long ago, I started graduate school as a roboticist. Researchers in mobile autonomous robotics hold a unique place in the Artificial Intelligent community; roboticists are responsible for pointing out that most of the things other researchers assume to be trivial are, in fact, excruciatingly difficult. Using physical sensors (e.g. cameras, sonar and touch sensors), it is only possible to recognize classes of objects by making use of strong constraints (e.g. absence of visual or physical clutter uniqueness of colors)–grounding the symbols reasoned about by classical planning systems is often impossible. Using physical actuators, seemingly simple tasks like moving forward a fixed distance in a straight line or picking up an object are made challenging due to the various imprecisions introduced by factors like slippage and assorted motor failures. Unless a robot's domain is heavily engineered, it is simply not possible for it to directly execute the sorts of plans produced by classical state-based planners–these plans abstract away important execution details that are needed to transition between the plan states.

All of these difficulties have driven robotics researchers to produce their own plan

representations and system architectures. One popular approach to plan representation is that of making reactive plans [29, 38] in which the appropriate course of action to take to accomplish a goal depends on the robot's situational context. Such plans allow robots to be robust in the face of many sensor and actuator failures and also to adapt to (or take advantage of) unforeseen changes in the environment. Typically, such plan representations are used by an upper level of a multi-level system architecture. The lower levels of the system serve to provide levels of abstraction, buffering the higher levels from some of the messiness of the physical world.

Unfortunately, it seems difficult to build robotic systems using these techniques in such a way that they also interact naturally with people. This is largely due to the way that roboticists tend to view the world as an adversary, posing difficulties that the robot must somehow overcome. Even in a seemingly interactive task like serving as a museum guide [13], the robot itself is mostly just trying to move from point to point without hitting anyone or anything. Many robotics researchers see interaction with people as the next important step for their projects. The robot competition and exhibition at the American Association for Artificial Intelligence conferences have featured robotic hors d'oeuvre servers [41] and the current Challenge Task requires a robot to participate in the conference (autonomously signing in and giving a brief presentation) [7]. Currently, the robot waiters' interactions tend toward the simplistic and cute, and it will probably be several years before any robots accomplish significant portions of the challenge. But, substantial research effort is being directed toward interactive robotics.

My thesis work began as such a research effort. Our laboratory robot, Chip (see Chapter 2), was skilled at locating small pieces of trash, picking them up, and sorting

them into a trash bin and a recycling bin [32]. He would also allow people to point at soda cans, either to be placed in a recycling bin [48] or to be hand-delivered to the pointer [37]. These two tasks demonstrated a limited interaction between a robot and a human, but also pushed the limits of what our robotic architecture supported. Using our approaches, it was relatively easy to use a simple interaction to tell the robot to initiate a particular task (e.g. directing it where to move [54]), but substantially more difficult to allow the robot and human to interact in interesting ways during the course of a task. Furthermore, in mobile autonomous robotics, it is hard to come up with many tasks where the robot interacts with people in interesting ways but which are also not too far beyond the state of current robotics research. (The RoboCup [52] is an example of an excellent domain for researching a wide range of robot/robot interactions.)

The Intelligent Classroom was selected as a domain that provides an interesting level of interaction and also exhibits many of the interesting challenges of mobile autonomous robotics research. As its user, the speaker, goes about his presentation, the Classroom watches and listens to him and, when appropriate, assists him. For example, the Classroom films the presentations that occur within it, producing a video feed that would be suitable for distance learning or for storing in presentation archives. To produce a useful video feed (i.e. to be a good assistant) the Classroom must capture all the activities in the Classroom that the speaker would wish his viewers to see. So, when the speaker is writing on the board, the Classroom must make sure that it focuses its presentation camera on the words and figures that the speaker is writing. It must zoom the camera in tight enough that the words can be read, but also capture the surrounding words so that people viewing the video feed

see the words in their greater context. Also, when the speaker is standing still and lecturing, the Classroom should zoom the camera in on his face so that the viewers can see his face expressions. To produce a reasonable video feed of a presentation, the Classroom must utilize a wealth of camera framing techniques, and in order to determine which technique is appropriate, the Classroom must have at least a superficial understanding of what the speaker is doing at any given moment. The Classroom assists by doing all the behind-the-scenes things that make a speaker's presentation go smoothly: controlling the various audio/visual components that the speaker needs and producing a video tape of the presentation.

When the speaker is lecturing from a set of slides, the Classroom must listen to the speaker so it can tell when it is time to switch slides. The speaker might indicate that he wishes to go on by issuing a command–saying, "Next slide, please" or touching the next slide "button" on the display screen. Or, as the Classroom follows along with what the speaker is saying, it can infer that the speaker wishes to go on to the next slide when he has finished talking about the current slide and has begun discussing a new one.

When the speaker wants the Classroom to play a particular video segment, the Classroom must cue up the video on the VCR, set the display to accept the VCR output, start the video, and begin listening for the various commands that are associated with playing videos (i.e. "pause", "skip forward" and "go back"). The speaker might express his desire to play the video by explicitly telling the Classroom to play the video segment, or, if the speaker provides a script of the presentation to the Classroom, it will know when it is the appropriate time to play the video.

Figures 1.1 and 1.2 show where the Intelligent Classroom domain lies in the space

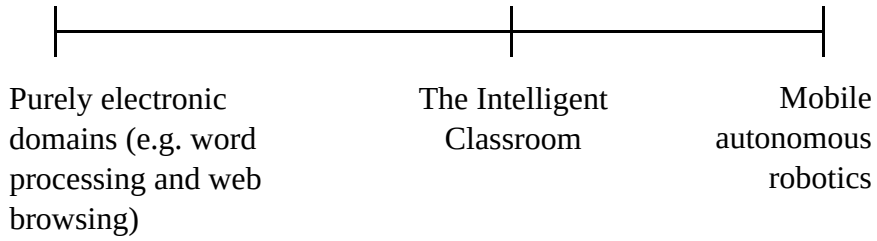


Figure 1.1: A graphic representation of the space of possible environments, considering the physical nature of the domain

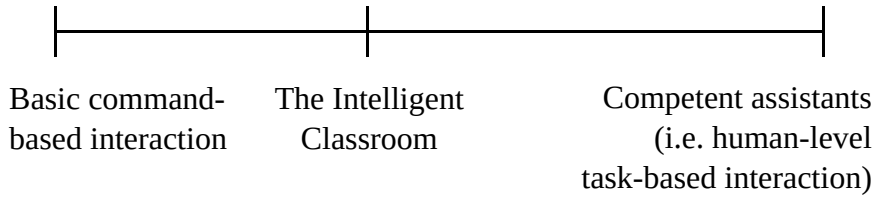


Figure 1.2: A graphic representation of the space of possible levels of interaction

of types of environment and interaction. In the environmental space, the Classroom lies between purely electronic domains and mobile autonomous robotic domains, slightly closer to the robotic end of the spectrum. The Classroom must use physical sensors (e.g. cameras and microphones) to sense what the speaker is doing, but the sensing camera is stationary. The Classroom's actuators are much easier to control than those of a robot; only the presentation camera requires continuous control and all of the components (e.g. cameras, VCRs and projectors) are commercial products with reliable control APIs. In the interaction space, the Classroom lies between that of purely command-based systems and truly natural human/human interaction, lying nearer to the simpler end. The Classroom interacts with its user during the course of his tasks, using the task context to understand his actions. This allows a

substantially more complicated level of interaction than can be accomplished using any purely command-based approach, but avoids many of the complicated requirements of a truly competent assistant (e.g. natural language interaction to control the interaction and deep reasoning about people's goals). The Intelligent Classroom's location in these spaces of environment and interaction makes it an ideal domain for my research: it is neither trivial nor impossibly difficult. The Intelligent Classroom provides a good domain for advancing the state of the art in human/robot interaction.

1.2 A process-based model of the world

The Intelligent Classroom adopts the Animate Agent Architecture's [31] way of acting in the world. Basically, in taking action, the agent sets some continuous actions in motion and then watches them run. While these continuous actions run, the agent watches for certain events to occur, prompting it to set some different continuous actions in motion. Each of these sets of continuous actions can be viewed as a step in doing the task. In this dissertation, I use the term *process* to refer to sequences of these steps. The steps in these processes may be totally ordered (with one step always leading directly to the next) or they may be only partially ordered (with the results of each step determining which step will follow). Furthermore, in the Classroom, these processes can be sequences of actions that the Classroom takes or sequences of actions that the Classroom observes the speaker take.

Consider the process a robot might use in crossing a room. It may set the continuous actions of (1) tracking its progress towards its goal, (2) moving towards its goal while using sonar to avoid stationary obstacles, and (3) visually watching for large moving objects that need to be actively avoided. The robot would be waiting

for events like (1) arriving at the goal (indicating that the robot can go on to do its next task), (2) being blocked from the goal by stationary obstacles (indicating that the robot should try a different technique for navigating), or (3) being on a collision course with a moving object (indicating that the robot must take evasive action). This description of a robot's process for crossing a room has steps for the general navigation (just described), for dealing with being blocked and for taking evasive action. The continuous actions are all a part of the first step of this process.

In the Classroom, we have extended this technique to include monitoring what other agents are doing. In situations where it knows what the speaker is doing, the Classroom will monitor the speaker as he goes about his task. The Classroom has a process-based representation of what the speaker is doing, and assumes that this representation adequately describes what the speaker will actually do. To monitor a speaker's process, the Classroom observes the running of the process, looking for the same events that the speaker is, and then observing the running of the appropriate next step when an event occurs. Most of the tasks that we expect speakers to perform in their presentations can be naturally represented using processes.

To gain some intuition as to how this dual nature of process description can be used, consider how people communicate process-like tasks to one another:

> To purchase a Coke from a Coke machine, you begin by inserting the appropriate coins in the coin slot. With each coin you insert, you should listen to hear whether the machine accepted it. When the machine accepts a coin, you will hear a pleasing deep "thunk" noise deep within the machine. When the machine rejects a coin, you will hear a jarring "clank" noise from the coin return. (Then you will have to try reinserting that

> coin or a inserting a substitute.) Once you have inserted the proper coins, then you press the button for the beverage you desire. You should hear a "thump" as the can is dropped into the dispensing area. Then you reach in and extract your beverage.

This description of how to use a Coke machine can, of course, be used as a plan for purchasing a can of Coke from a vending machine. But, it can also be used in understanding what someone else does when he is using a Coke machine. For example, if he has just inserted a coin and you hear the "clank" noise, you understand that he reaches towards the coin return in order to retrieve the rejected coin. In executing a process, an agent actually takes the actions described in the steps and waits for the appropriate events. In monitoring a process that a different agent is executing, an agent observes the step actions but waits for the same events that it would if it were executing the process itself.

Now we have sufficient background to formally define some terms:

- An *agent* is anything (person, machine or other object) that can be viewed as causing change in the environment. Agents implicitly have some sort of internal state that is not directly observable. For people, intention is an important part of this internal state. For machines, this internal state corresponds to the hidden state of the machine that is relevant to its operation. For instance, a Coke machine might be broken or might already have some money inserted.

- A *process* is a sequence of actions (or compound actions) that will be executed by a single agent. For example, the sequence of actions in a STRIPS plan [28] for stacking blocks would be considered a process. This definition of process

separates the actions from the goals they are intended to achieve. In the section about plan recognition (Section 1.3), I show how these processes can be tied together with goals to make plans. We have developed a language for describing processes that are hierarchical and have simultaneous actions and conditional branching.

Given our commitment to using processes as our representation of all the activities going on in our environment, there are several research issues that must be looked at. The next few sections look at some of the interesting ones that are addressed in this dissertation.

1.2.1 Tracking a process' progress

How can you tell whether a process is progressing well? Sometimes there is a measurable quantity that you can monitor while the process is running. For example, when you are hammering a nail into a piece of wood, you can observe how much of the nail is sticking out of the wood and how far the nail is straying from being perpendicular to the plane of the wood. While hammering, you expect that the nail should gradually descend into the wood and, generally, that the nail should point straight out from the wood. If the nail stops descending, something has gone wrong; perhaps the nail is lodged in a knot. If the nail leans too far, you risk bending over the head of the nail. In both cases, the process is not progressing well–if no changes are made, the process will ultimately fail.

In a process where there are no physical quantities that are easy to measure, there is nearly always a range of time that you would expect the activity to take. In the hammering example, you would probably wonder if the wood is rotten if the

```
(define-process (move-to-board-and-write)
  (main-actor
   (person ?speaker))
  (roles
   (chalkboard ?board)
   (chalk ?chalk))
  (steps
   (_go (do _move (move-to ?board))
        (track _delta (track-floor-distance ?speaker ?board))
        (wait-for (_move :done) _2)
        (ensure _delta (:decreasing)))
   ...)
  (time-constraints
   (process-duration (range 30 3000) (expected 300))
   (step-duration _go (range 0 30) (expected 5))
   ...))
```

Figure 1.3: A partial process definition for the speaker going to the chalkboard and writing

nail went all the way in with one swing of the hammer. But, you would also know that something is wrong if the nail has not gone all the way in to the wood after ten minutes of pounding. By combining information about physically measurable quantities and time, we can detect many situations where processes are not running properly.

Figure 1.3 shows a portion of the representation used in the Intelligent Classroom for the speaker's process of going to the board and writing. The first line defines how the process will be invoked; the process is named `move-to-board-and-write` and takes no arguments. The next two clauses define the participants in the process. The `main-actor` refers to the agent who is executing the process and `(person ?speaker)` specifies that this agent must be a person and he will be referred to using the variable `?speaker` throughout the remainder of the process definition. The `roles` clause describes other agents and objects that are important to the execution of the process.

The `steps` clause defines the steps that make up the process. (In the figure, only

one step is shown.) Each step is given a label (a symbol starting with an underscore), which is used to refer to the step throughout the process definition. Following the step label, the step definition describes two sub-processes in that step. The first (labeled `_move`) is for doing the actual moving toward the chalkboard and the second (labeled `_delta`) is for tracking how far the speaker is from the board. Then, the step definition indicates that we should wait for the `_move` sub-process to indicate that it is done and then go on to the step labeled `_2` in the process. The final clause in the step definition specifies that the distance between the speaker and the chalkboard should be decreasing.

Finally, the `time-constraints` clause specifies how long the various parts of the process should take. The `process-duration` clause says that the `move-to-board--and-write` process should take between 30 and 3000 seconds and can typically be expected to take 300 seconds. The `step-duration` clause says that the first step of the process should take up to 30 seconds but will typically take only 5 seconds.

So, when someone executes this process, the definition indicates that initially he will be moving towards the board and that he will arrive there in 30 seconds or less. If he moves away from the board or just takes too long, then we know that his task of going to the board and writing is not proceeding well. The process definition does not provide any information about why the process is not proceeding well–perhaps there is something preventing the speaker from moving to the board; perhaps he was temporarily distracted and will go to the board in a moment; or perhaps he never intended to go to the board and write at all. With what the Classroom can observe, it usually has no way of knowing which of these situations actually holds; there is simply not enough information. Fortunately, for the Classroom's tasks, it only needs

to know that a process is not progressing well–not the particular reason.[1]

1.2.2 The notion of perspective in tracking processes

In the last section I discussed some techniques for determining if a process is progressing well. It is important to note that, in both examples, each of the quantities can be measured both by an agent executing the process and by an agent that is observing the process. This is crucially important in allowing agents to use the same process representation for processes that they will execute and for processes that they need to observe. If we are to represent a big part of the state of the environment as the set of processes being executed in it, it is important that we are able to represent these processes in a uniform way. Aside from implementational and representational concerns, this ability also is pleasing from an introspective stand-point–we, as people, are able to recognize and follow when people do things the same way we do and are also able to observe people as they do something and then go and do it ourselves. The process representation discussed in this dissertation is designed to facilitate utilizing this dual nature of processes.

The processes that the Intelligent Classroom concerns itself with fall into two classes: processes that the Classroom itself is executing and processes that it is observing other agents execute. Because it is tracking the progress of both kinds of processes, the Classroom only needs to distinguish between the classes of processes in dealing with the actions in the processes. In its own processes, of course, it performs the actions specified in the process. For other processes, the Classroom decides

[1] As implemented, the Intelligent Classroom is interested in what activities the speaker performs and uses this information to decide how to cooperate with his tasks. If a process is not progressing well, the Classroom assumes that this is the result of the speaker doing something else. This assumption greatly simplifies the plan recognition process, but unfortunately makes it impossible for the Classroom to assist the speaker when his plans are failing.

```
(define-process (move-to-board-and-write)
  (main-actor
   (person ?speaker))
  (roles
   (chalkboard ?board)
   (chalk ?chalk))
  (steps
   (_go (do _move (move-to ?board))
        (track _delta (track-floor-distance ?speaker ?board))
        (wait-for (_move :done) _do)
        (ensure _delta (:decreasing)))
   (_do (do _choose (select-chalk ?chalk))
        (wait-for (_choose :done) _3))
   (_3 (do _pickup (pick-up-from ?chalk ?board))
       (wait-for (holding ?speaker ?chalk) _4))
   (_4 (do _write (write-on-board ?board))
       (wait-for (_write :done) :done)))
  (time-constraints
   (process-duration (range 30 3000) (expected 300))
   (step-duration _go (range 0 30) (expected 5))
   (step-duration _do (range 0 1) (expected 0))
   (step-duration _3 (range 0 5))
   (step-duration _4 (range 5 3000) (expected 300))))
```

Figure 1.4: A process definition for the speaker going to the chalkboard and writing

what to do based on the particular action. For an action that serves to spawn a sub-process (as in the `_move` step in the process definition from the previous section), the Classroom will spawn the sub-process and then observe it. For an action that it knows how to observe (the Classroom has a process that is specifically designed for observing that type of action), the Classroom will execute the process to observe that action. For all other actions, the Classroom will simply ignore the action and rely on other clues from the process that it is observing.

Figure 1.4 shows a more complete representation for the speaker's process of going to the board and writing. From the perspective of the speaker, this process tells the steps involved in going to the board and writing something. This task involves moving to the board, selecting a piece of chalk to write with, picking it up, and, finally, writing with it for a while. Presumably, the speaker will have processes defined for moving,

selecting, picking up and writing, which he will execute at the appropriate times.

In observing the speaker execute this process, the Classroom will first observe the speaker performing the first step. It will spawn the sub-process for `(move-to ?board)` which will allow the Classroom to observe some particular details of the speaker's moving. At the same time, the Classroom will begin tracking the distance between the speaker and the chalkboard and keeping track of how long the speaker is taking to get there. When the speaker arrives at the board, the `_move` sub-process will signal that it is done and the Classroom will advance the process to the second step.

The Classroom has no way of monitoring the speaker's decision process in selecting a writing implement. Because there are also no physical quantities to measure, the Classroom relies on the expected completion time for the step to guess when the next step should begin. In this example, the step is expected to take no time at all, so the Classroom immediately advances the process to the third step.

For the third step, the Classroom will again execute a sub-process (this time for `pick-up-from`) that will allow it to observe the speaker. This sub-process gives the details of reaching down to pick up the piece of chalk from the base of the board. It specifies the appropriate measurable quantities and will indicate when the sub-process has completed.

In the fourth step, the Classroom has a process specifically designed for observing the speaker writing (not shown in the figure) and so spawns this observation process. This process details what it looks like when somebody is writing: how the arm is positioned; how the board is changed; and how to tell when the writer is done. When the observation process indicates the writing is done, the Classroom then knows that

the process itself is completed.

This example is intended to demonstrate how the representation for a particular process can be used from two perspectives. From the perspective of an actor (the speaker), it tells what to do. From the perspective of an observer (the Classroom), it tells what it looks like when someone else is doing it. Both the actor and the observer utilize the progress conditions and the mechanisms for dealing with the hierarchical decomposition of the process are very similar. In looking at how this dual nature of processes is used in the Intelligent Classroom, it is important to state that I do not actually believe that speakers literally represent their plans in this way and execute them using these mechanisms. Rather, I believe that speakers have process representations that are analogous to these and that the Classroom can observe them as if they actually were using the processes it represents.

1.2.3 Keeping track of lots of processes

As mentioned earlier, an important part of the Classroom's understanding of the world is a set of all the processes that it is keeping track of. Due to the hierarchical decomposition of processes and the fact that the Classroom may be involved in many tasks simultaneously, the Classroom often needs to keep track of dozens of processes. The Classroom relies on its Process Manager to monitor the progress of all these processes.

Figure 1.5 shows the important structures in the Process Manager. At the highest level (on the left), there is a set of all the active processes. In each process, the steps are treated as the states of a finite automaton; to monitor a process, the Process Manager keeps track of which step the process is in. The Process Manager will advance a process to a different state when it observes one of the events that the

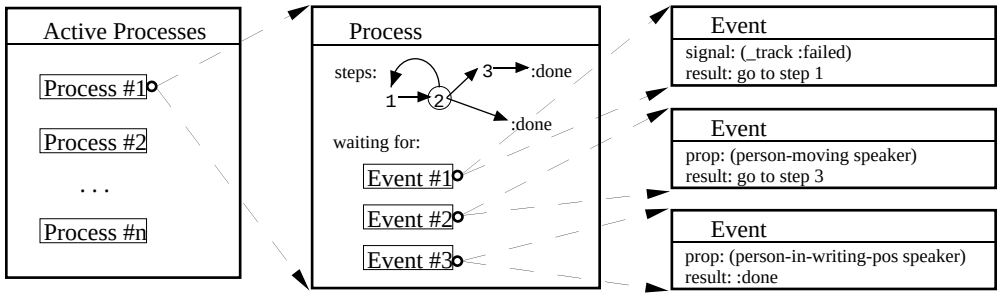


Figure 1.5: Important structures in the Process Manager

process is waiting for. For example, the process is currently in step _2 and if the process designated by `_track` signals failure, the process will go on to step _1.

When a process moves to a new step, the Process Manager spawns processes for each of the actions given in the step, adding them to the set of active processes. Then, while these sub-processes run, the original process waits for any of the events given in the step definition to occur. Finally, when such an event occurs, the process manager halts all of the sub-processes it spawned for this step and advances the process to a new step based on which event occurred.

There are two types of events that a process may wait on: an event being signaled by another process and a memory proposition becoming true. All of the events discussed in the previous sections are of the first type. (The second type will be detailed later in this dissertation.) Processes send signals when they start, when the finish, and sometimes to communicate important events they have observed. The Process Manager is responsible for keeping track of which processes are waiting for which events (and memory propositions) and making sure that when an event occurs, the appropriate processes are advanced.

1.3 Plan recognition as a prerequisite for cooperation

With the Process Manager, the Intelligent Classroom has the mechanisms that it needs to follow along with what the speaker does–if it knows what the speaker is doing. So, without some mechanism for inferring what the speaker is doing, the Classroom would need the speaker to provide a detailed description of everything that he will be doing in his presentation and then need the speaker to actually do everything he said he would. It is unreasonable to demand a speaker to do this, so the Classroom needs to be able to recognize what process(es) the speaker is executing, based on what it observes him do. In the Artificial Intelligence literature this mechanism is generally called plan recognition. With the terms used in this dissertation, it would be more accurate to call this process recognition. However, since the processes the Classroom recognizes are all have closely associated plans (I will formally define plan later), I will use the term plan recognition for both uses.

When a speaker is doing a presentation, the Classroom tries to keep its representation of what the speaker is doing consistent with what it observes the speaker do. In the Process Manager, this means continually stepping through its set of processes to keep them synchronized with the speaker and occasionally revising the set of processes when necessary. The basic idea is that, when the Classroom observes the speaker take some action, there should be a process in the Process Manager that "explains" why he took that action. For instance, if the Classroom knows that the speaker is going over to the board to write, the speaker's action of picking up a piece of chalk is explained because that was what the Classroom expects him to do after arriving at the board. However, if instead of writing, the speaker walks away from

the board, the walking action is not explained–apparently the speaker is not writing on the board.

In situations where a speaker's action is not explained by any of the processes in the Process Manager, the Classroom must revise its understanding of what the speaker is doing; it must add a process to the process manager that explains the speaker's action. Such a process must include the speaker's action in one of its steps. Initially, there may be several processes that adequately explain the speaker's single action, even though the speaker is only performing one of them. But, as the speaker continues in what he is doing, the Classroom will be able to rule out the incorrect ones.

At this point it is important to note that the Classroom's plan recognition differs significantly from most plan recognition research: the Classroom is much more concerned with *what* the speaker is doing than *why* he is doing it. This is because the Classroom directly uses the inferred plans to produce cooperative behavior, rather than reasoning about the speaker's goals in order to select an appropriate course of action. Also, the Classroom performs plan recognition in a robotic domain, using the readings from physical sensors, rather than interpreting natural language or other high-level action representations. For both of these reasons, the plan recognition techniques described in this dissertation do not do the goal-level reasoning that is described in much of the plan recognition literature. Given the Classroom's task and domain constraints, such reasoning is unnecessary and would be exceedingly difficult, if not impossible, to employ.

1.3.1 The basic plan recognition algorithm

When faced with an unexplained action, the Classroom finds all the processes in its library that: (1) can have the speaker as their main actors, (2) have the unexplained action as a part of one of their steps, and (3) are consistent with the speaker's previous actions. The first two conditions are easily verified by looking through the process definitions and looking for the appropriate information. The third condition requires a little more work. If the unexplained action is in the process' first step, this condition trivially holds because there are not any actions before the step that could be inconsistent with what actually was observed. For other cases, the Classroom must simulate the running of the process to be sure that everything the process would expect to happen before the unexplained action, did happen. To facilitate this, the Classroom maintains a history of the speaker's last several actions. For a particular process under consideration, the Classroom enumerates all the sequences of steps that lead up to unexplained action, collecting all the speaker actions that the steps predict. (While for processes with many steps this may be infeasible, for the relatively short processes with limited branching used in the Classroom, this works quite well.) Then, the Classroom looks through the history and sees if any of the action sequences are contained in it. If so, the process is considered a candidate explanation for the speaker's action–it has met all three conditions.

Once the Classroom has a set of candidate explanations for the speaker's action, it must reduce the set to the single correct explanation. This is not immediately possible because the Classroom simply does not have enough information to determine which process the speaker is performing. So, the Classroom eliminates the incorrect processes as the speaker performs more actions. There are two ways that the

Classroom is able to eliminate the incorrect processes. For the first, the Classroom eliminates processes that are not progressing properly. This means either that one of the time constraints was broken or that one of the physically measurable quantities is not behaving properly. For instance, if the Classroom has hypothesized that the speaker is going over to write on the board, it expects the speaker to get there within a few seconds and to be moving generally towards the board. If the speaker stops before getting to the board or starts moving away from the board, the Classroom will be able to eliminate the `move-to-board-and-write` process from consideration. In both cases, the temporal constraint will be violated and the measurable quantity (the distance between the speaker and the board) will not be behaving properly.

The second way that a process can be eliminated is if the speaker performs actions that are inconsistent with him executing the process. Since the Classroom assumes that the speaker is only doing one thing at a time, the Classroom expects all the speaker's actions to be explained by the speaker's process. Therefore, if the speaker takes an action that is not a part of a candidate process, that process can be eliminated from consideration. This is much like the situation that prompted the plan recognition in the first place: the Classroom thought the speaker was doing one thing, but the speaker's actions indicated that he was doing something else.

The Process Manager provides the functionality for keeping track of all the candidate explanations and eliminating them as future speaker actions contradict them. When the Classroom observes the speaker taking an action, it tells the Process Manager to deal with the event of the action occurring. Then, the Process Manager tries to use this event to advance the processes in its process set. In doing so, it also makes sure that the top-level speaker process (or one of its sub-processes) explained

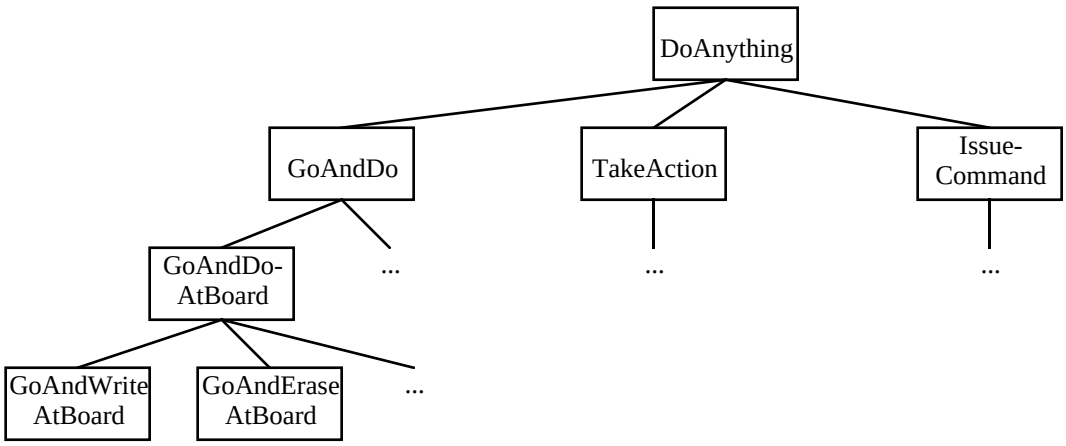


Figure 1.6: A portion of the Intelligent Classroom's speaker process hierarchy

the speaker event. If the speaker event is not explained, then the Classroom will need to find a new explanation for what the speaker is doing. It gathers a set of candidate explanations and adds these processes to the Process Manager, marking them as explanations.

As the Process Manager runs, it checks that each candidate process is progressing normally, and eliminates those that are not. When the Classroom observes the speaker taking another action, the Process Manager will advance the candidate explanations using the event associated with this action. Any of the processes that do not explain the action are removed from the Process Manager. When only one candidate explanation remains, it is accepted as the correct explanation and its mark as a candidate explanation is removed.

1.3.2 Dealing with ambiguity

One tricky bit: the Classroom acts based on its understanding of what the speaker is doing, and often must act before it has eliminated the candidate explanations. To

deal with this, we have organized all the speaker processes into a single ISA hierarchy, and the Classroom always acts upon a generalization of its set of candidate processes. The leaves of the hierarchy are the "real" processes. For example, the Classroom has processes for going to the board to write, going to the board to revise something, and going to the board to point at something. A generalization of these three processes is a process for going the board to do something. A further generalization is a process for going somewhere to do something. Figure 1.6 shows a portion of the speaker process hierarchy used in the Classroom. For any set of candidate processes, there is a unique most-specific generalization process. For a widely diverse set of candidate processes, the generalization process may be the (most general) "do-anything" process, which uses a conservative camera technique during the Classroom's time of uncertainty.

The generalization (interior) processes capture the common structure of their children. The various "chalkboard" processes are broken into "go to the board" and "do something" parts. As the generalization process is changed (as candidate explanations are rejected), the common structure allows the Process Manager to start the more specific generalization process at the right point.

Whenever the Classroom tries to explain a speaker's action, it adds a generalization process to the Process Manager, along with the candidate explanations. This generalization process is considered to be what the speaker is actually doing. Initially, it will be an overly vague notion of what he is doing, but at all times, it is as precise as the evidence warrants. Whenever the Classroom is able to further specify this generalization process (as candidate explanations are rejected), the Process Manager replaces the process with a more specific one. The new process is started at the same step as the old process was in, and all of the appropriate variable bindings are copied

over as well. This allows the new process to pick up as close as possible to where the old one left off.

1.3.3 How plan recognition fits into the Classroom

Plan recognition is fundamental in supporting the Intelligent Classroom's "world view"–that it can provide an explanation for everything that it senses. When things are going as expected, no plan recognition is needed. But when something unexpected happens, the Classroom uses plan recognition to build a new explanation for what is going on.

The Intelligent Classroom has a very particular use for plan recognition: figuring out what the speaker is doing so that the Classroom knows what it ought to do to cooperate. The speaker might frequently change what he is doing, so the Classroom must act quickly and often has little time to determine precisely what he is doing. Fortunately, in the plans that the Classroom needs to recognize, it is nearly always appropriate to choose actions based on an overly general notion of what the speaker is doing. The Classroom can be more helpful with a precise understanding, but it is still helpful with a more general one.

As an example, consider what happens when the speaker starts walking toward the board. There are many possible explanations for his action. Among them, he might intend to: write on the board, revise what's on the board, erase something from the board, or point at something on the board. Initially, the Classroom's generalization of these candidate explanations is that the speaker intends to go to the board to do something. So, in deciding how to cooperate with the speaker, the Classroom will initially use this explanation. Then, if the speaker picks up the eraser, the Classroom will be able to reject the candidate processes that deal with writing and pointing,

leaving the erasing explanation as sole candidate explanation. Now the Classroom's generalization process has converged to what the speaker is actually doing–the plan recognition is complete.

1.4 Cooperation

Given its understanding of what is going on, how does the Intelligent Classroom decide what to do? First, given the top-level process the speaker is executing, the Classroom selects the (unique) plan associated with it. Then, for activities where the speaker is expecting the Classroom to cooperate with his intentions, the Classroom must execute its process(es) in the speaker's plan. The important thing here is to be sure not only to do the right thing, but also to do it at the right time. The plans define "at the right time" by stating how steps in different processes need to be synchronized. For example, a plan might essentially say that the Classroom should change the camera mode immediately after the speaker enters the chalkboard area.

1.4.1 The structure of plans

Formally, I define *plan* as a set of processes (often to be executed by a number of different agents) that when run together successfully, accomplish some goal. In the Intelligent Classroom, many plans have some processes executed by the speaker and other processes executed by the Classroom. It is important to note that this is not really a new definition of plan–any plan that has a step of the form "wait for this event to happen" is implicitly representing processes external to its main actor. This definition makes explicit the presence of other agents or exogenous events.

Figure 1.7 shows a portion of the plan representation the Classroom uses when

```
(define-plan (move-to-board-and-write)
  (main-actor
   (person ?speaker))
  (roles
   (intelligent-classroom ?classroom))
  (accomplishes
   (?speaker
    (do (move-to-board-and-write))))
  (processes
   (_p1 ?speaker
        (move-to-board-and-write))
   (_p2 ?classroom
        (film-move-to-board-and-write
          ?speaker)))
  (synchronization
   (starts (_p1 _go) (_p2 _go))
   (equals (_p1 _4) (_p2 _do))))
```

```
(define-process (move-to-board-and-write)
  (main-actor
   (person ?speaker))
  (roles
   (chalkboard ?board)
   (chalk ?chalk))
  (steps
   (_go (do _move (move-to ?board))
        (track _delta (track-floor-distance ?speaker ?board))
        (wait-for (_move :done) _do)
        (ensure _delta (:decreasing)))
   (_do (do _choose (select-chalk ?chalk))
        (wait-for (_choose :done) _3))
   (_3 (do _pickup (pick-up-from ?chalk ?board))
       (wait-for (holding ?speaker ?chalk) _4))
   (_4 (do _write (write-on-board ?board))
       (wait-for (_write :done) :done)))
  (time-constraints
   (process-duration (range 30 3000) (expected 300))
   (step-duration _go (range 0 30) (expected 5))
   (step-duration _do (range 0 1) (expected 0))
   (step-duration _3 (range 0 5))
   (step-duration _4 (range 5 3000) (expected 300))))

(define-process (film-move-to-board-and-write ?speaker)
  (main-actor
   (intelligent-classroom ?classroom))
  (roles
   (person ?speaker))
  (steps
   (_go (do _film1 (film-moving-speaker ?speaker))
        (wait-for (_film1 :done) _do))
   (_do (do _film2 (film-writing-speaker ?speaker))
        (wait-for (_film2 :done) :done))))
```

Figure 1.7: Plan and process definitions for the speaker going to the chalkboard and writing

the speaker walks over to the board and writes. The plan includes the previously described process for what the speaker does (walking and writing) and a process which details how the Classroom should film the speaker as he walks and writes. Finally, the plan definition itself ties these processes to a particular goal and specifies how the processes fit together.

Like the process definition, the plan definition specifies who the participants are in the plan (the speaker and the Classroom). The plan then specifies what it is intended to accomplish. In this example, the plan is intended to be used when the speaker wishes to move to the board and write something. This is not a goal in the strict

sense of achieving a particular memory proposition (though the plan representation does allow such goals). `Do` goals (as opposed to `achieve` goals) are used in situations where it is difficult to express the goal in terms of a proposition to be achieved. For example, it is hard to come up with a proposition to associate with the plan for taking a walk around the block. We have found that many of the tasks a speaker typically performs during a lecture are difficult in this way.

After the statement of the goal of the plan, the definition shows what processes should be executed as a part of the plan. Each process is given a label, a main actor and its invocation (specifying which process to execute and with what arguments). So, the second item in processes states that the plan requires a process labeled `_p2`, executed by the Classroom, named `film-move-to-board-and-write`, and with the speaker as an argument.

Finally, the plan details how the two processes must be synchronized. It states that the first steps of both processes start at the same time and that the last steps of both processes take place over the same interval of time. This way of specifying how the processes fit together makes the synchronization explicit, rather than mixing the synchronization into the Classroom's filming process.

1.4.2 Structure of the process manager

Figure 1.8 shows a block diagram of the key software components of the Intelligent Classroom. At the center is the Process Manager. It holds the process-based model of the world and does all the work of understanding what is going on. All the other components just give the Process Manager the information it needs.

Below the Process Manager is the interface with the physical world. This component gives the Process Manager the sensory data it needs: speaker actions and

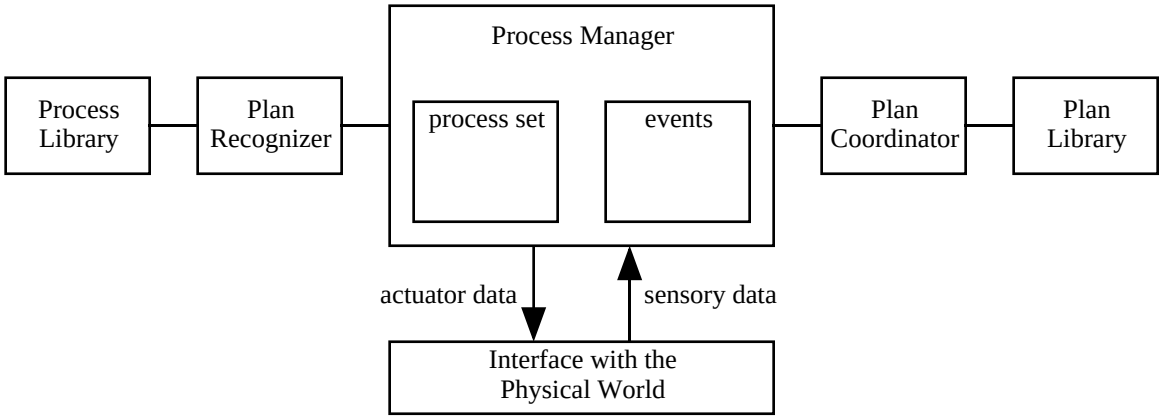


Figure 1.8: A block diagram of the key software components of the Intelligent Classroom)

requested information about the behavior of measurable physical quantities. The sensory data is used as events to advance the processes in the Process Manager. In turn, the Process Manager tells this interface component what actions to take and how to configure the sensory system.

To the left of the Process Manager is the plan recognition component. It is responsible for proposing candidate processes and selecting a generalization process for them. The Process Manager tells this component when the current process set failed to explain a speaker action. It gives the plan recognition component what the action was and what speaker actions preceded it (the event history). The plan recognition component uses this information to select a set of candidate explanations (from its library of processes), which it then tells the Process Manager to enable, along with a generalization process. As candidate processes are ruled out, the Process Manager informs the plan recognition component, often causing it to revise the generalization process.

To the right of the Process Manager is the plan coordination component. It is responsible for finding the appropriate plan for what the speaker is doing and for making sure that the Classroom's actions are properly synchronized with those of the speaker. Whenever the generalization process (for describing what the speaker is currently doing) changes, the Process Manager tells the plan coordination component, prompting it to locate the plan in its library that tells how to cooperate with the speaker in what he is doing. This plan will include a set of processes, which the component tells the Process Manager to enable. This plan also includes the information about how the processes must be synchronized. The synchronization is enforced in the Process Manager by having the appropriate processes wait for the events of particular process steps starting or ending.

1.5 Jabberwocky

In the implementation of the Intelligent Classroom, we have duplicated many of the sorts of interactions that would be considered typical between a speaker and a human audio/visual coordinator (e.g. playing video tapes or performing camera-work). But in designing our slide controller, we saw an opportunity to significantly improve how speakers give slide-based presentations. Rather than providing just a new means of advancing to the next slide, Jabberwocky [36] (the resulting slide controller) allows a speaker to control his slide presentation by discussing the slides he wants shown. Jabberwocky matches the words and phrases the speaker says to words and phrases in the slides to determine which slide the speaker is discussing at any moment.

Jabberwocky uses the process-based techniques of the Classroom, but extends them through the use of probability and some simple learning. We are optimistic

that the approaches we use to address the problems of poor speech recognition in Jabberwocky will also prove useful in deal with the lesser problems of sensor noise in the Classroom.

1.6 Where to now?

This section has served to introduce the key ideas behind my dissertation research. The remainder of the dissertation serves to flesh out many of the technical details of implementing them in a real system. Also, it looks at some additional areas of interest which were not looked at in this introduction. Since this introduction provides a broad overview of the research, there are few dependencies between later chapters. As a result, readers are encouraged to skip forward to whatever sections are of interest to them and to skip over those that are not.

Chapter 2 focuses on the techniques the Intelligent Classroom uses to sense and act in the physical world. It outlines the various pieces of research that became a part of the Classroom's implementation. It shows how the earlier work on robotics evolved as we tried to make the robot more cooperative. Finally, it provides insight into what goes on in the low-level implementation of the Classroom in observing the speaker as he goes about his presentation.

Chapter 3 looks at the process-based representation in great detail: how physical processes are represented and how the Classroom uses the representation to take action and to observe the speaker as he takes action.

Chapter 4 gets at the heart of this research: cooperation. It describes the Classroom's plan representation and how it tells the Classroom how best to cooperate with the speaker. Furthermore, the chapter details the plan recognition algorithm, which

uses the functionality of the Process Manager to determine what the speaker is doing. It also tells how the Classroom uses a generalization technique to take action before it is certain precisely what the speaker is doing.

Chapter 5 discusses the Jabberwocky slide-controller. In particular, it looks at the ways that it fits in with the process-based techniques used in the Classroom and at the ways that it differs, through the introduction of probability and learning.

Chapter 6 relates the research in this dissertation to research in the many areas that the Classroom draws from: particularly intelligent environments and plan recognition.

Finally, Chapter 7 concludes this dissertation by addressing some potential complaints that are not dealt with elsewhere in the dissertation, discussing a few of the key philosophical ideas behind this research, and suggesting some future research directions.

Chapter 2

Dealing with the physical world

Kristian Hammond, in discussing this research, likes to proclaim that the Intelligent Classroom is our solution to one of the major navigation problems facing robotics researchers. While the robotics community tries to figure out how to keep their robots from running into the walls of their laboratories, we just made the walls be a part of our robot!

The rest of this dissertation assumes that the Intelligent Classroom is able to sense (at a low level) what the speaker is doing and is also able to take action by controlling its various automated components. This chapter discusses the Classroom's low-level interaction with the physical world and how this interaction fits in with the whole system. The techniques used in the Classroom are adapted from techniques we[1] used in earlier mobile robotics research, because the Classroom must deal with many of the same issues as a mobile robot. It must interpret noisy results from physical sensors, act through physical actuators, respond to changes in the world in real time, and exhibit a certain degree of autonomy. The first portion of this chapter is devoted

[1]This section discusses the research of the Animate Agent Project at the University of Chicago. So, in this section "we" is used to refer to the researchers who worked on the project: Jim Firby, Josh Flachsbart, David Franklin, Ari Kahn, Peter Prokopowicz and Michael Swain.

Figure 2.1: Chip at the University of Chicago

to providing an overview of the robotics research and showing how it influenced the design of the Classroom. The remainder of this chapter looks at how the robotics research was adjusted for use in the Classroom.

In discussing the low-level techniques, it is important to note why they are important in the Classroom and in robotics research in general. Essentially, physical sensors do not produce the discrete events that the higher-level components require and the discrete commands produced by the higher-level do not map neatly into actuator settings. The two-layer architecture we use provides a convenient layer of abstraction, making the messy continuous world into a somewhat neater and more discrete one.

2.1 The Animate Agent Architecture

While we were designing the Intelligent Classroom, we had to deal with many of the same issues that mobile robotics researchers have to deal with. In fact, the concept of

the Intelligent Classroom came out of our existing work on physically interacting with our mobile robot, Chip (see Figure 2.1). As a result, it seemed natural to capitalize on the relevant portions of our robotics research when building the Classroom. The backbone of our robotics research, around which all our other work was built, was the Animate Agent Architecture [31]. The Animate Agent Architecture consists of a high-level discrete system and a low-level continuous system and commits to a clean interface between them. The high-level system tells the low-level system how it should be configured; the low-level system reports important events back to the high-level system.

On our robot, the high-level system was the RAP reactive task execution system [29]. Given a goal (such as "put all the cans in the room into the recycling bin" or "move over to the person"), the RAP system chooses a method for achieving the goal and executes it. A goal and its methods are stored in *reactive action packages* (RAPs). The RAP system holds a library of these RAPs, indexed by the goal they achieve. A RAP may contain many different methods for achieving a goal, each appropriate in a different context.

After trying a method for achieving a goal, the RAP system checks to see if it was successful. If so, all is well. If not, it will consider other methods for achieving the goal. Eventually, it will either succeed, or give up, determining that it cannot achieve the goal. Each method contains a plan for achieving the goal: a set of steps and information such as what order the steps need to executed in, which steps can be executed in parallel, and what conditions need to hold while particular steps are being executed. Each step specifies a subgoal, which is then recursively pursued. Finally, there are primitive RAPs, which interact with the low-level system. They enable sets

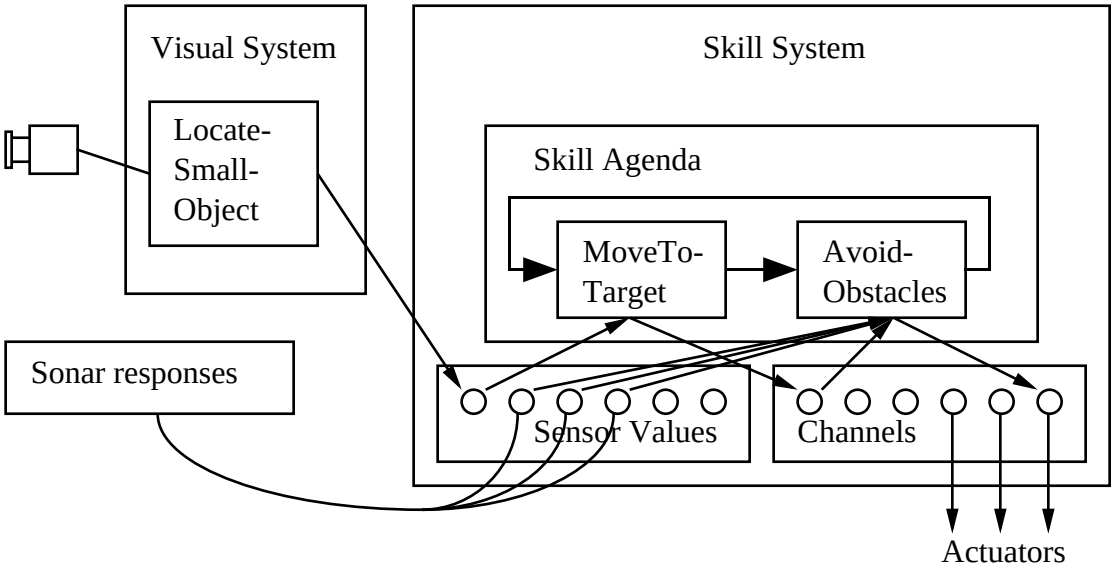


Figure 2.2: The skill system configuration as Chip heads towards a can

of skills and vision routines and specify what events to wait for, and what to do in response when one of those events occurs.

The low-level system on our robot was a skill-based system that linked together reactive skills and vision modules to form tight control loops. This system actually interacted with the physical world, processing the sensory inputs (such as camera images, sonar readings, and base and arm encoders), and controlling the actuators (i.e. moving the robot). Individual skills looked for potentially important events in their execution (usually transitions and boundary conditions) and reported them back to the RAP system. In the simplest kind of control loop, the results of processing a sensory input would be piped into a channel that would, in turn, be read by a skill that used those results to determine how to set an actuator. For example, when preparing to pick up a soda can from the floor, the robot had to precisely align itself

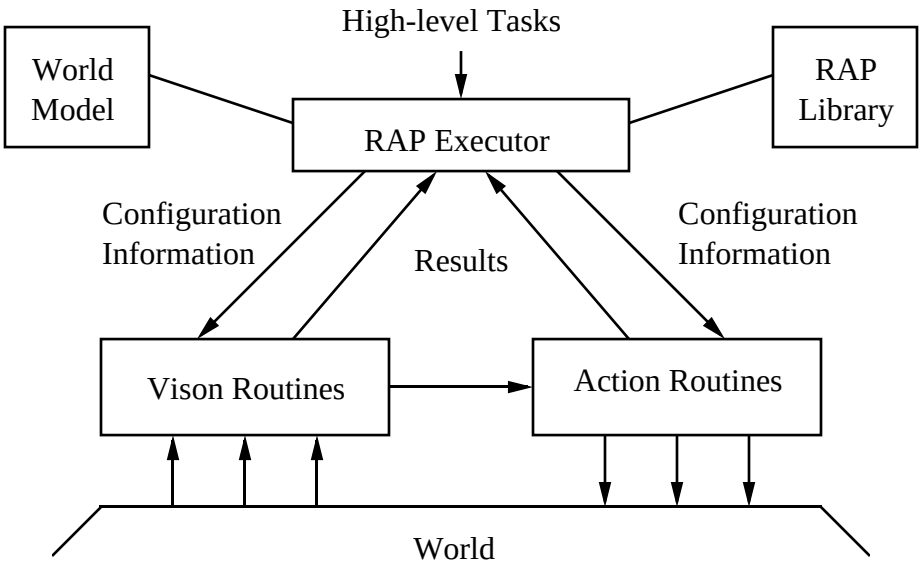


Figure 2.3: The Animate Agent Architecture

with the can. To do this, it enabled a vision routine that located the closest can in a camera image and enabled a skill that moved the robot towards a target location. The results from the vision routine were piped into the TargetCoords channel. This channel was read by the robot's MoveToTarget skill, causing the robot to move. The robot's motion changed the camera image, producing a new target location, once again causing the robot to move. Once the robot was aligned with the can, the MoveToTarget skill would signal the RAP system, telling it that it was aligned with its target. Figure 2.2 shows the set of skills and visual routines that were used in the primitive navigation of our robot.

One important element of the Animate Agent Architecture's commitment to a clean interface between its high- and low-level systems is that only the high-level system reasons about the agent's goals. The low-level system is always unaware

of what it is being used to accomplish. It reports all the discrete events that the high-level system might find useful, and leaves it to the high-level system to decide what to do about them. This encourages skill reuse; once we had written a skill that moved the robot forward until either it had moved a prescribed distance or its sonar indicated that its way was blocked, we did not have to write new versions for all the different types of navigation tasks we wanted the robot to perform. (This was important because the programs that actually dealt with the physical sensors and actuators were terribly difficult to get to work well.) The Animate Agent Architecture is graphically represented in Figure 2.3, showing the components' relationships to one another and the flow of control between them.

To provide an example of how the low-level system's lack of goal awareness is important, we look at two ways the skill for moving forward safely was used, and, in particular, how the robot dealt with being blocked from moving forward. When moving to a particular location in the room, the robot would aim towards the target location and then move forward the appropriate distance to arrive there. If its sonar indicated that something was in the way, the skill would tell the primitive RAP, which would then have to decide how to deviate from its original path to avoid the obstacle. In this case, the "blocked" event meant that its current method would not work, and the robot needed to try another one. When moving up to a person to deliver something, the robot would aim towards the person and then roll up to them. As before, if its sonar indicated that something was in the way, the skill would tell the primitive RAP, but, in this case, the RAP would know that the obstacle was the person. In this case, the "blocked" event meant that its current method had succeeded–the robot had successfully approached the person.

In the Intelligent Classroom, we have adopted the Animate Agent Architecture. The RAP system of the high-level has been replaced by a similar system that also provides better support for plan recognition and synchronized coordination between multiple agents. The system of skills and vision modules of the low-level has been expanded, but retains the important design philosophies. Finally, the clean interface between the systems is almost unchanged.

2.2 Perseus and Gargoyle

In order to physically interact with people successfully, an agent (e.g. a robot or the Intelligent Classroom) must be able to see them. It must be able to visually locate them and recognize their gestures. In developing the Animate Agent Architecture, we accomplished this first using the Perseus purposive visual system [47, 49]. With Perseus, our robot was able to visually track a person as he moved around in an otherwise static visual scene. It kept track of where he was standing and the positions of his head and hands. When the person assumed a pointing pose (i.e. he held his hand stationary and away from his body), it would visually trace out a ray from his hand and try to determine what he was pointing at [48]. Later, we utilized this ability in making Chip into a robot waiter [37]. In this task, the restaurant "patron" would have a number of soda cans on the floor around him. To show that he wanted a particular can, he would point at it. The robot would then move over to that can, pick it up, move over to the patron, and hand him the can.

The Perseus system is composed of several main components: feature maps (including edge, color and motion), object representations (including lights, floor, cans and people), markers (for cans, heads and hands), a short term memory that holds

the most recent segmentation map, a long term memory that associates symbols with object representations, and a library of visual routines. When Perseus is given a visual task, it selects the appropriate visual routine, which then instantiates the appropriate object representations, which then use the feature maps and short term visual memory to locate the objects and use markers to keep track of them. For example, the task of recognizing what soda can a person is pointing at utilizes a visual routine that first visually locates the person using the motion feature map and places a marker on him. Then it segments him (using the short term visual memory) and places markers on his head and hands and gives each marker an appropriate method for tracking its body part. Now Perseus monitors the positions of the head and hand markers; if either hand marker is held stationary out from body, this is understood as a pointing gesture. Perseus then computes the line of sight from the head and through the pointing hand and searches for a can along this line.

Concurrently with the work on Perseus, we explored ways of making the visual system interact more closely with the other elements of the Animate Agent Architecture. At the time, all the vision routines were separate programs, each used to sense a particular thing in a particular way. Also, in order to produce results in a timely fashion, these potentially large programs had to be resident in memory; and, due to their size, they often had to be run on separate computers. In our task of picking up pieces of trash and sorting them into trash and recycling containers [32], we had visual routines for: (1) locating all the pieces of trash in a camera image (at a range of 1-5 meters), (2) tracking the closest piece of trash while approaching it, (3) locating a trash can (or recycling bin) in a camera image, (4) tracking the trash can while approaching it. If four visual routines were needed for such a simple task,

in a static domain, we knew that even slightly more complex tasks would require substantially more routines. This would require more memory and also the tedium of writing perhaps dozens of nearly identical visual routines, each to be used in slightly different circumstances.

This work led to the design of Gargoyle [67]. Gargoyle provides an architecture in which the computer vision system for an agent can easily adapt to changes in its context. This context consists of the particular task that the agent is performing, the state of the world, and the configuration of the agent (in particular, the position and zoom of its camera). Gargoyle's underlying idea is that all of these elements of context can be used to render the vision system more efficient and more accurate. For example, in the task of filming a speaker as he walks around and lectures, the Classroom needs to track the position of the speaker's head fairly accurately, but can pay less attention to the rest of his body. Also, if the speaker is known to be wearing clothes whose color is significantly different than the background, the Classroom can use computationally inexpensive color methods to track him. Finally, if the Classroom knows that its sensing camera is stationary, it can use background-subtraction methods to determine the speaker's position. All three of these types of contextual information provide visual constraints that the vision system can utilize to be more effective; Gargoyle allows the visual system to be dynamically reconfigured to make use of these constraints.

Gargoyle is able to accomplish this due to its modular nature. Given a visual task, Gargoyle builds a pipeline of vision modules to accomplish that task. For example, one pipeline for visually tracking a speaker as he moves around in the Classroom includes a background subtraction module, an edge detector, a leg finder, and a

morphological segmenter. Each module is chosen based both on its functionality and on what constraints it relies on. In the example pipeline, the background subtraction module relies on a stationary background and the leg finder requires that the walls not have very much visual texture (i.e. be painted in a single, solid color, rather than having a bold pattern printed on them). Furthermore, if changes in the environment alter which constraints hold, Gargoyle can dynamically swap in modules that utilize the new set of constraints. For many modules (or sets of modules) there are other modules that provide similar functionality but that differ in when they are most applicable. Gargoyle can use the most applicable ones, at the moments that they are most applicable.

In the Intelligent Classroom, we are continuing the development of the Gargoyle vision system as we use it to observe the speaker as he moves about. Many of the tracking algorithms used in Perseus have been adapted for Gargoyle [34] and we have developed new techniques, such as icon recognition [35] to deal with particular Classroom situations.

2.3 The Skill Manager

Through the development of Gargoyle, we found that the modules in Gargoyle began to look more and more like the skills. Like many modules in the skill-based system, each Gargoyle module inputs some data from one of the sensors or from another module, does some computation on the data, and then passes the results on to the next module. Also, the behavior of the Gargoyle modules can be modified while they are running by adjusting their parameters. Because of these similarities, we have decided to make the interfaces to the skills and visual routines identical from the

Process Manager's perspective. In fact, through the rest of this dissertation, I will not make any distinction at all between visual routines and what had previously been called skills–I will use the terms skills and low-level routines to refer to both.

The Skill Manager keeps track of all the active skills and facilitates their communication with the Process Manager. Recall that the purpose of the skill system is to provide a layer of abstraction to connect the messy continuous world to a discrete one that can be understood by the processes. The skills are connected together into control loops which: (1) continuously poll the physical sensors, usually observing the sensed values' behavior over time, passing results on to other skills, and reporting relevant events that it observes (e.g. "the person has started moving" or "the camera image is suddenly too dark") and (2) continuously set the physical actuators, usually based on results from other skills.

When a running process requires that a skill be started, the Process Manager asks the Skill Manager to start it and tells it whether (and how) it should be connected to other skills. The Skill Manager then determines what physical computer the skill should be run on (currently the computer vision skills are run on one computer and all the other skills are run on another), sets up communications channels between it and any other skills that it is connected to, sets up a communications channel to deal with event reporting, and sets it running. Similarly, when the running process no longer requires that skill, it informs the Skill Manager and the skill is removed.

In the Intelligent Classroom, there are certain sets of skills where each set provides a useful piece of functionality and the skills are frequently enabled as a group. Furthermore, the information produced by some groups, such as the set of processes that visually track the speaker's motion through the Classroom, is often needed by

more than one process. So, the Skill Manager provides a method for enabling named sets of skills, thereby simplifying how the processes start skills and helping to ensure that the skills are not duplicating the efforts of other skills. These named sets, called *pipelines*[2] (after the similar structure in Gargoyle) detail what skills make up the set, how they should be connected, and what other pipelines the pipeline depends on–a pipeline can build on the results of another one. In the Classroom, we utilize pipelines that observe the speaker's motion, look for writing actions and look for gestures related to giving a slide presentation.

2.4 Overall Intelligent Classroom architecture

The Intelligent Classroom uses the two-layer architecture of the Animate Agent Architecture. At the bottom, we have a skill system linked together with Gargoyle (controlled by the Skill Manager), and at the top, we have the process-based execution system discussed in the remainder of this dissertation. Figure 2.4 shows the overall architecture of the Classroom.

There are a few notable differences between this architecture and that shown in Figure 2.3:

- In the Classroom, the skills (including Gargoyle modules) are linked together in a slightly different way than they were in the Animate Agent Architecture.

[2]A pipeline (in the computer processor design sense) makes use of several specialized processing units running in parallel, each doing a small portion of the computation for a single instruction and passing the results on to the next processing unit in the pipeline. So, while a non-pipelined processor leaves many of its parts idle most of the time, a pipelined processor improves the utilization of its parts, often dramatically increasing its throughput. The "pipelines" in the Skill Manager and Gargoyle run on one and two processor machines, and so do not actually achieve this great parallelization . . . yet.

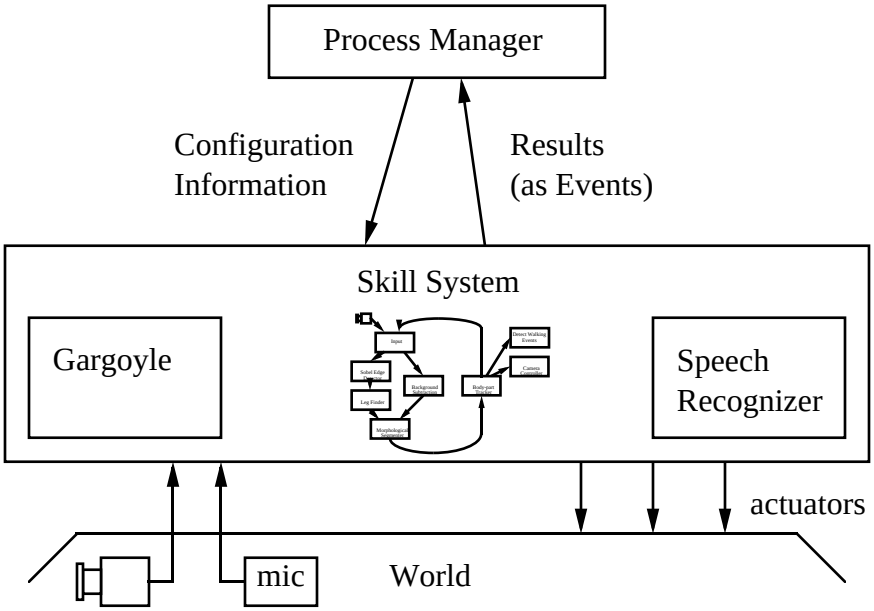


Figure 2.4: The Intelligent Classroom's Architecture

While the Animate Agent Architecture utilized globally named channels to connect skills together, the skills (and vision modules) in the Classroom have explicit inputs and outputs, which the Classroom connects together. This allows the Classroom to enable multiple instances of a given skill (e.g. a hand tracker).

- The Classroom has a different set of sensors and actuators. The Classroom senses its world through a fixed camera (positioned to cover the entire front wall of the Classroom), a microphone connected to a commercial speech recognizer (IBM's ViaVoice software [46]), and information about its presentation camera (its current position and zoom). The Classroom controls a presentation camera (filming the presentation), a VCR, a digital projector, a speech generator, and (virtually) the lights in the Classroom.

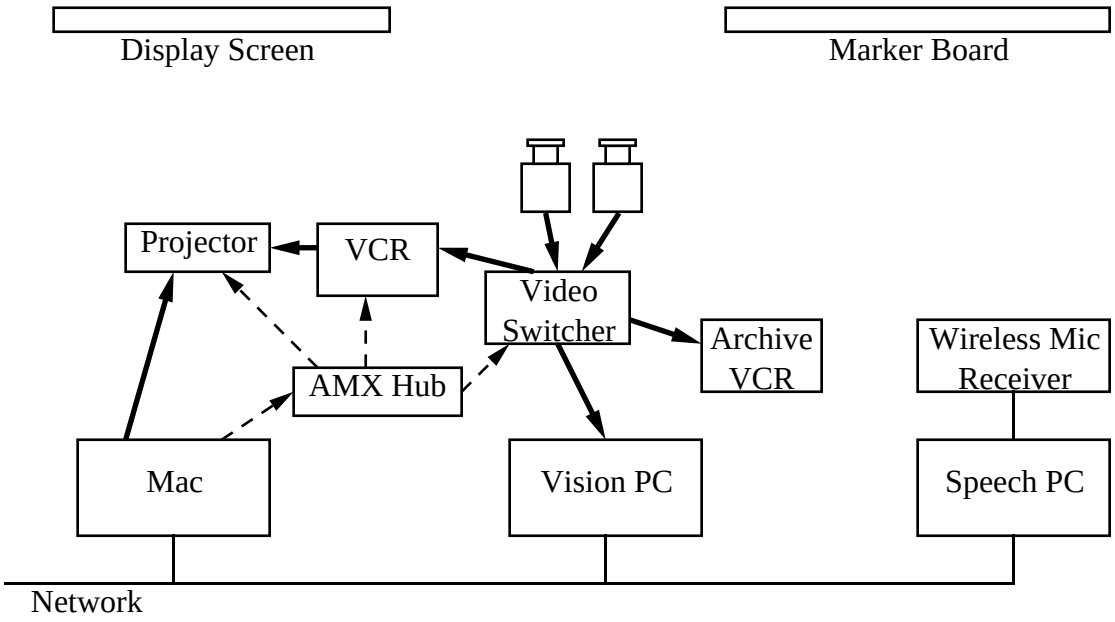


Figure 2.5: The Intelligent Classroom's Hardware Components. The thick arrows indicate the flow of video signals and the thin dashed arrows indicate the flow of serial control data.

- In the Classroom, the Process Manager may dynamically configure vision routines by altering their run-time parameters. In the Animate Agent Architecture this is not possible due to the nature of the vision routines. This difference is particularly important in that it allows the vision system to rapidly adapt to changes in its visual context.

Figure 2.5 shows all of the computers and other hardware components that make up the implementation of the Intelligent Classroom. Starting at the bottom of the figure, the Classroom is built around a network of three computers: a Macintosh computer running the Process Manager and all the non-vision-based skills, a PC running Gargoyle, and another PC running the speech recognizer. The three computers

communicate using TCP streams through the network. The Macintosh controls the various video components of the Classroom, using an AMX AXCENT3 Hub, which we essentially use as a serial port splitter. These video components include an InFocus LP725 digital projector, a VCR that is used to play video segments during a presentation, and an Extron SW6 video switcher which controls what camera source is sent to Gargoyle and to the VCR that is recording the presentation. The Classroom utilizes two cameras: a stationary Panasonic WV-BP120 (for most of the computer vision processing) and a pan-tilt-zoom Sony EVI-D30 (the common presentation camera). Finally, the Classroom uses a Shure wireless microphone set.

2.5 An example: Filming a speaker as he walks about

Now we examine what happens as a speaker walks about in the Intelligent Classroom while lecturing. For the purpose of this example, we will focus on what happens at the lower level of the architecture, glossing over the details of what is actually happening at the higher level. Initially, immediately after the Classroom discovers that speaker has begun to move around, the Classroom sends a series of instructions to the skill system to set it in the configuration shown in Figure 2.6. On the left are the Gargoyle modules performing the computer vision processing, and on the right are the skills that use the results of the vision processing. The MorphologicalSegmenter module produces camera coordinates for the speaker's head, hands, center and feet. These coordinates are then converted into global (Classroom-based) coordinates by the BodyPartTracker skill and passed on to the CameraController and DetectWalkingEvents skills.

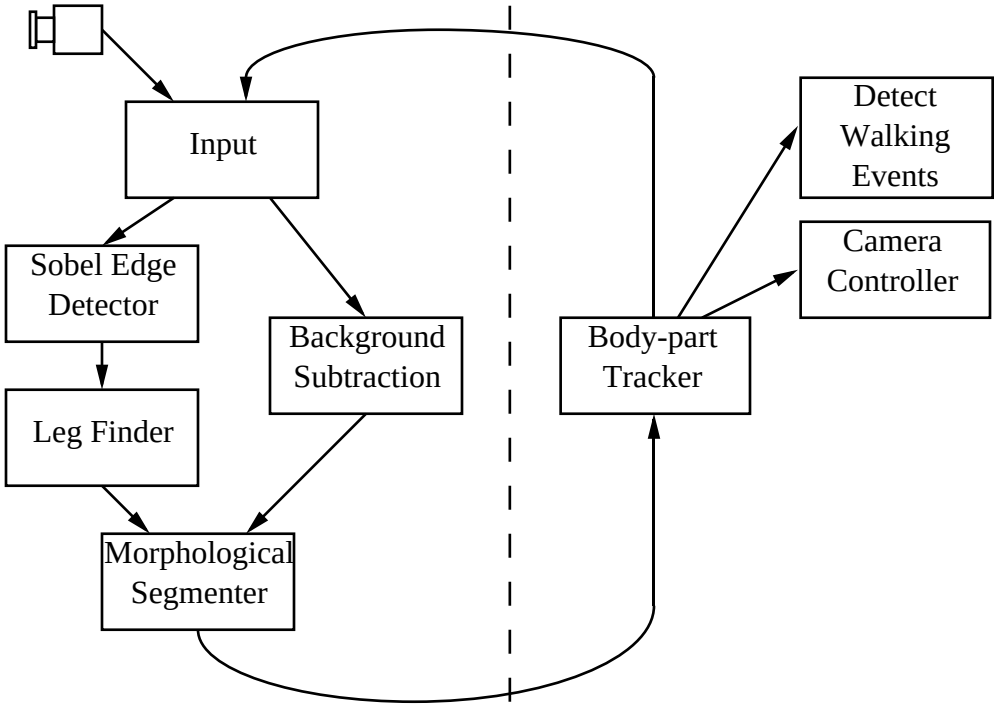


Figure 2.6: The Gargoyle modules (on the left) and other skills (on the right) used as the speaker walks around

Initially, the CameraController skill is configured for using a rather wide camera angle as the speaker moves about. This is important because, while the vision system is able to process about five camera frames a second, there is a system latency of about one half a second from the moment the speaker moves to the moment that the camera moves to follow him. Even with the wide camera angle, if a speaker moves back and forth quickly, the camera has trouble keeping him in the frame. Figure 2.7 shows a snapshot of the state of the low-level system as the speaker is walking. On the left is the segmentation image produced by Gargoyle. In the middle is the Classroom's representation of the speaker (and important objects and spaces in the

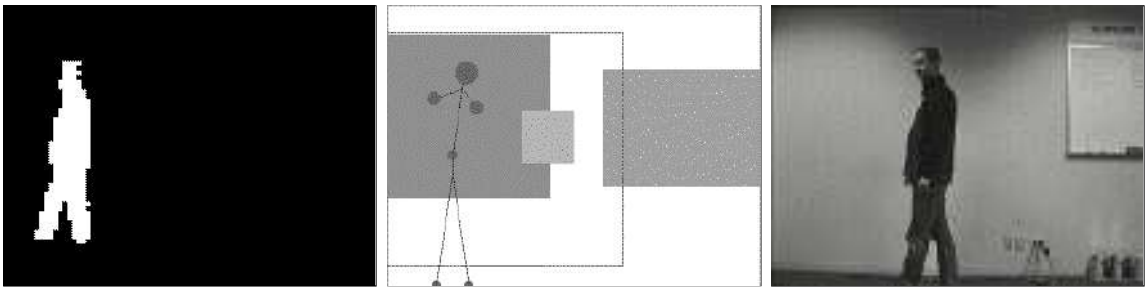


Figure 2.7: Segmentation image, Classroom representation and presentation image for when the speaker is walking.

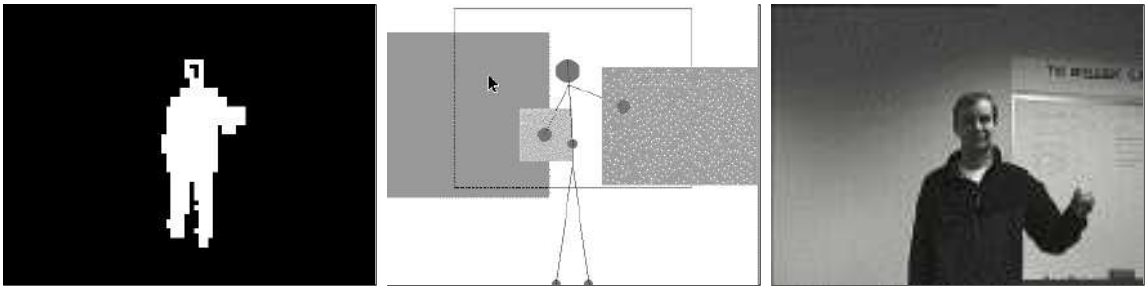


Figure 2.8: Segmentation image, Classroom representation and presentation image for when the speaker is standing.

Classroom), which is used to direct the presentation camera, resulting in the camera image on the right.

While the speaker is moving around, the DetectWalkingEvents skill is looking for walking events to report to the higher-level system. This skill detects when the speaker approaches an important region of the Classroom, such as the marker board or the projector screen; it also detects when the speaker starts or stops moving.[3] For

[3]The Intelligent Classroom tracks the speaker's position on the floor using the camera coordinates of his head. The x-coordinate can be used to compute what the speaker is standing in front of because the camera's position and the Classroom's layout are both fixed. The y-coordinate can be used to compute the speaker's approximate distance from the wall, if we know the speaker's height. (As the

this example, we will only consider what happens when the speaker starts and stops moving. After the speaker has remained stationary for two camera frames (about half of a second), the DetectWalkingEvents skill determines that he has stopped moving and reports this event to the higher-level system. This happens to be an event that the Classroom was expecting (in one of its active processes) and, as a result of the event, it reconfigures the CameraController skill to use a much tighter camera technique, zooming in on the speaker's face and upper torso. Figure 2.8 shows a snapshot of the state of the low-level system as the speaker is standing and speaking. If the DetectWalkingEvents skill later determines that the speaker has resumed walking (i.e. the speaker has moved significantly for two consecutive camera frames), it will report this to the higher-level system, resulting in the skill system returning to the original configuration.

2.6 Summary

The Intelligent Classroom owes much of its design to its evolution from our earlier robotics research. The overall architecture is based upon the Animate Agent Architecture, which separates the reactive low-level components from the more planful high-level ones. The basic algorithms for visually tracking were adapted from the Perseus visual system for use in the still-evolving Gargoyle system. The Classroom's skill system, though implemented quite differently from that used in the robotics research, functions almost identically. The major departure from the Animate Agent Architecture is the replacement of the RAPs system with the plan and process system described in the remainder of this dissertation.

speaker moves away from the wall, he appears taller in the camera image.)

As a whole, the architectural design of the Classroom is quite mature (being the product of many researchers, working nearly a decade) and, as a result, is unlikely to change substantially as this research continues. In the immediate future, we will be working on more ways of applying the speaker's task context to aid the vision system. This task will test the clean interface between the high and low-level systems. Most other low-level work in the Classroom will involve adding new skills and Gargoyle modules to provide more acting and sensing capabilities.

Chapter 3

Dealing with processes

In order to decide how to cooperate, the Intelligent Classroom relies heavily on its understanding of what is going on in the Classroom. As discussed in the Introduction, much of this understanding is contained in a representation of what processes are currently active. This representation is managed by the Process Manager. In this chapter, we look at the process representation used in the Classroom: the important components and the key algorithms for tracking the progress of an active process. In addition, we look in depth at two processes in the Classroom's process library that utilize important aspects of the representation and that also serve as crucial components in the examples in the next chapter.

The algorithms in this chapter assume that there is a lower-level system to handle the physical interaction with the world. This system needs to convert the actual sensor input values into discrete events that the Process Manager reasons about. Also, it needs to convert the commands from the Process Manager into actuator settings. Our implementation of such a system for the Classroom is discussed briefly in the previous chapter.

3.1 The philosophy and representation of processes

The Intelligent Classroom tries to understand the world around it by explaining its observations in terms of a set of processes that are being executed by the various agents there. As defined in the Introduction, an *agent* is a person or machine that can be viewed as causing change in the world, and a *process* is a sequence of actions that will be executed by a single agent. So, the Classroom's understanding of the activity in its environment is the set of processes being executed by the agents in the environment. Generally, in the Classroom, there are two agents to consider: the speaker and the Classroom itself. The processes being executed are the speaker's activities (such as writing on the board, doing a slide presentation, and walking around and lecturing) and the Classroom's activities (such as controlling the presentation camera, playing a video tape, and switching a slide).

Before we look at the way processes are represented in the Classroom, we should look at the three ways that processes are used in the Classroom. First, the Classroom executes processes in service of particular goals. For instance, if the Classroom has a goal of playing a segment of a video tape, it will execute a process in service of that goal; this process will cue the tape, set the appropriate video input for the projector, adjust the lights and play the segment. Second, the Classroom matches the speaker's actions against the actions specified by the processes it knows about, in order to infer the speaker's goals. For instance, if the speaker walks up to the marker board and picks up the eraser, the Classroom can infer that he intends to erase something from the board. Depending on what the Classroom infers, it may also determine that there is some course of action it can take to cooperate. Third, when the Classroom knows what the speaker is doing, it can guess what he is likely to do next by looking at future

actions in the speaker's process. The process representation used in the Classroom is designed to facilitate all three of these uses of processes.

3.1.1 Process definitions

In the Intelligent Classroom, processes are represented as sequences of steps involving continuous actions and waiting for particular discrete events. Each step in a process definition describes a set of actions (generally the running of processes or low-level skills) that make up the continuous activity for that step. The step then describes the events that are important in that step. These events include the conditions reported by skills, the starting and stopping of sub-processes, and events associated with the behavior of measurable values (e.g. "the speaker is no longer getting closer to the board"). Finally, the step specifies how each event affects the progress of the process. Some events may cause the process to halt (either successfully or in failure); others may advance it to a new step. Still others may not affect the running of the process at all and instead adjust the parameters for a running skill or make an assertion to memory.

Figure 3.1 shows a process definition for the speaker going to the chalkboard and writing. The definition can be divided into five parts:

- The *invocation* defines how this process is to be invoked as a sub-process or as one of the processes in a plan. The first item in the invocation (`move-to--board-and-write` in the example process) is the name given to the process. Any other items in the invocation define the variables that serve as arguments to the process. The variable binding process will be discussed later.

- The *main-actor definition* describes what kind of agent can execute this process

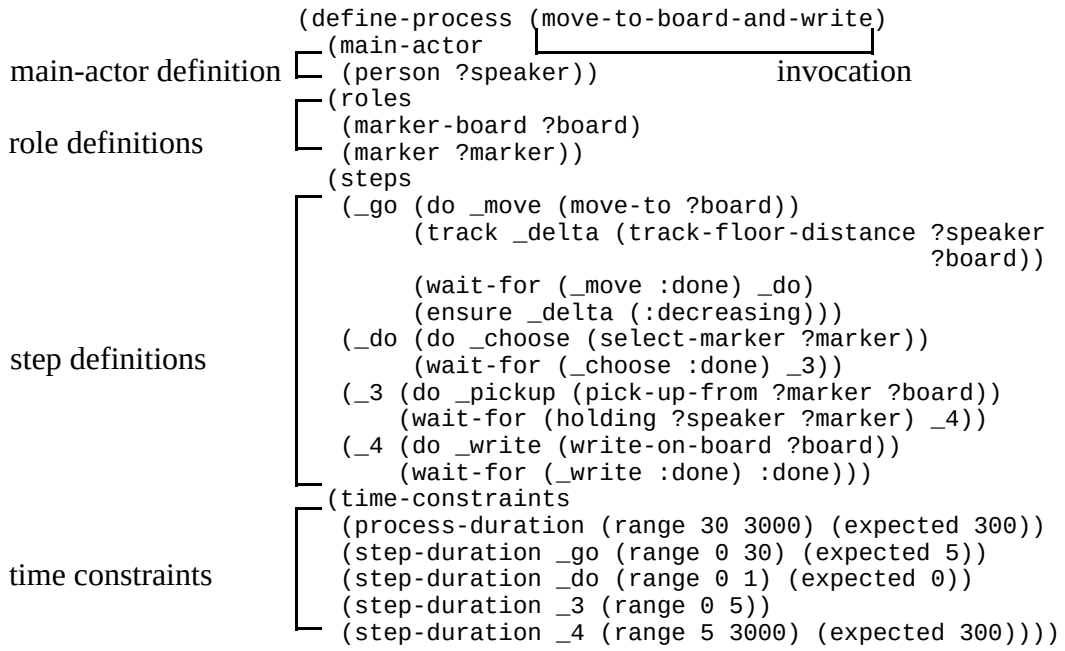


Figure 3.1: A process definition for the speaker going to the marker board and writing

and defines a variable with which to refer to this agent within the process definition. In the example definition, `(person ?speaker)` is a short hand description indicating that the agent must be an instance of a sub-class of `person` and will be referred to as `?speaker`.

- The *role definitions* describe the various agents and objects that play an important part in the execution of the process. Each role definition describes what can fill that role and defines a variable for referring to the role filler. These descriptions will be discussed in the next section. Role definitions are often used to describe the arguments given in the process invocation.

- The *step definitions* describe all the steps that make up the process and define

how to advance from step to step when executing it. The first item in each step definition is the identifier given to that step. After the step identifier are one or more action definitions, defining the continuous actions that are to run during the duration of the step. Finally, there are one or more event handlers, defining the important events that may occur during the execution of the step, and describing what to do when those events do happen. The first step definition defines the first step that will be executed as a part of this process, but the order that the remaining steps are executed in depends on what events occur.

- The *time constraints* define how long the process (or steps within the process) should take. Each time constraint first specifies an interval of time (`process--duration` indicates the running of the entire process and `step-duration _1` indicates the running of the step labeled `_1`), then gives a range of possible durations in seconds, and, optionally, includes an expected duration.

In addition to these parts, a process definition may contain `set-up` and `clean-up` clauses. A `set-up` clause defines a set of actions that can be viewed as a step that takes place during the entire duration of the process. Often, a `set-up` clause is used to start up sensing skills or pipelines that produce results used by all of the process' steps. A `clean-up` clause defines actions that effectively put the Classroom back into a safe state after the process is done. This cleaning up often involves reversing memory assertions made in the `set-up` clause or in the other steps.

3.1.2 Descriptions

Sometimes, in describing an object, simply specifying a class for the object is insufficient. For instance, when selecting a pen for writing with, rather just choosing

```
(described-by ?speaker {instance-of c-person})

(described-by ?marker {instance-of c-marker
                       (slots (color i-blue))
                       (relations (has-ink ?this))}
```

Figure 3.2: Two examples of using descriptions: "a person" and "a blue pen that has not run out of ink". (The descriptions are enclosed in curly braces.)

an object of class pen, the speaker may want "a blue pen that has not run out of ink." To support this kind of description, we use a representational approach based on [79]. The basic idea is that there are descriptions of objects and then there are actual grounded objects. A description specifies the sorts of objects that satisfy it. There may be hundreds of descriptions that fit a given object, and a given description may be satisfied by hundreds of different objects. An object can be represented as an identifier tied to the most specific description of the object.

Figure 3.2 shows two example descriptions. Each description is made up of three parts: class description, relational modifiers, and general conditions. The class description defines what kinds of objects satisfy the description. In the first description, only instances of the class `c-person` (or some sub-class of `c-person`) will satisfy the description. The relational modifiers specify relation-value pairs that must hold for the object. In the second description, the relational modifier indicates that only objects with a color of `i-blue` will satisfy the description. In most cases, the relations will be on object attributes (such as color), but the description representation does not require this. (For example, you could use a relational modifier to specify that the described person's mother is a doctor.) Finally, the general conditions list memory propositions that must hold for the described object. The special variable `?this` is

used to refer to the described object in the propositions. In the second description, only objects that have not run out of ink will satisfy the description.

In the process representation, a description can always be used in place of an instance or class identifier: in process invocations, role definitions and event descriptions. In using a description, rather than a particular instance, a process is rendered more flexible. For example, if the `marker` role in a "write on the board" process is filled with a description, the agent executing the process need not commit to a particular marker until it is actually necessary. Once the agent has decided which marker to use, the marker description is refined to specify this particular marker.

Currently in the Classroom these descriptions are not utilized to their full expressive potential; much of the time, it is sufficient to simply specify a class that an object must be an instance of. However, there are many places where the descriptions are utilized. In addition, when we add more complicated natural language understanding, we expect that the descriptions will prove invaluable [33]. In the short term, the descriptions do allow the Classroom to detect problems; if the speaker picks up a pen that the Classroom knows is out of ink, the Classroom can anticipate the writing problem because that pen does not satisfy the description of an appropriate pen for writing with.

Throughout this dissertation I use the convention of adding a "`i-`" prefix to all identifiers referring to grounded instances and adding a "`c-`" prefix to all identifiers referring to classes of objects. In addition, descriptions are enclosed in curly-braces, indicating that they are distinct entities, rather than predicate logic clauses.

3.1.3 Variable binding

An important part of the state of an active process is its set of variable bindings. These bindings hold the arguments that the process was invoked with, the agents and objects filling the process roles, and other bindings accumulated during its execution. A binding may refer to a particular grounded object or provide a description of the sort of object that may fill the binding. As a process executes, it may add new bindings or refine existing bindings. The basic binding process is based upon the binding techniques used in the RAP system [30] and that in turn is derived from the variable binding techniques used in such programming languages as Prolog. However, the description refinement process warrants further explanation.

A description can be thought of as defining a set of objects in the world that satisfy it. (In the Classroom, we never enumerate these sets, but it is useful to think about descriptions this way in understanding the refinement process.) A description may be refined by merging a second description with the first or by specifying a particular grounded object (which has its own attached description). In the first case, the set of objects satisfying the refined description will be the intersection of the sets of objects satisfying the two descriptions. In the second case, the set of objects will contain the single object–if it satisfies the description. In either case, if the set is empty, the refinement is deemed a failure and, as a result, the binding itself fails. Otherwise, the variable is bound to this refined description.

Figure 3.3 gives the basic algorithm used to merge two descriptions. Each step involves adding either the result of merging corresponding pieces of two descriptions, or the pieces themselves, if there is no corresponding piece. Two class constraints can be merged if one of the specified classes is a subclass of the other; the resulting class

MergeDescriptions(Desc1, Desc2) -> Desc3

1. Merge Classes: Set the class of Desc3 to the more specific of the classes of Desc1 and Desc2. If they are incompatible, fail.
2. Merge Slots: For each slot in Desc1 or Desc2 that is unique to that description, add the slot to Desc3. For each slot shared by Desc1 and Desc2, add a slot to Desc3 that is the result of merging the slots (i.e. recursively call MergeDescriptions on the two slot descriptions).
3. Merge Clauses: Add all the clauses from Desc1 and Desc2 to Desc3. Remove duplicate clauses.

Figure 3.3: The basic description-merging algorithm. Desc3 is the result of merging Desc1 and Desc2.

constraint will be the more specific of the two. If one description has a slot that the other lacks, that slot will be added to the merged description.

Figure 3.4 shows a portion of a process definition for dealing with the speaker walking over to and writing on one of the marker boards in the Classroom. The boxes on the right represent the bindings set during the execution of the process. Initially, the only bound variables are those found in the process invocation and roles. Furthermore, the bindings for the `marker-board` role initially does not specify which board he will be writing on–it only specifies that it will be a marker-board. As the process progresses, the bindings for the `marker-board` and `marker` roles will each be refined in turn.

3.1.4 Process actions

Actions produce change in the Classroom. These changes may take place in the distinct areas listed below. Figure 3.5 gives examples of each type of action.

```
(define-process (move-to-board-and-write)
  (main-actor
   (person ?speaker))
  (roles
   (markerboard ?board)
   (marker ?marker))
  (steps
   (_go (do _move (move-to ?board))
        (track _delta
               (track-floor-distance ?speaker ?board))
        (wait-for (_move :done) _do)
        (ensure _delta (:decreasing)))
   (_do (do _choose (select-marker ?marker))
        (wait-for (_choose :done) _3))
   (_3 (do _pickup (pick-up-from ?marker ?board))
       (wait-for (holding ?speaker ?marker) _4))
   (_4 (do _write (write-on-board ?board))
       (wait-for (_write :done) :done)))
  (time-constraints
   (process-duration (range 30 3000) (expected 300))
   (step-duration _go (range 0 30) (expected 5))
   (step-duration _do (range 0 1) (expected 0))
   (step-duration _3 (range 0 5))
   (step-duration _4 (range 5 3000) (expected 300))))
```

```
((?speaker i-dave)
 (?board {instance-of c-board})
 (?marker {instance-of c-marker}))

((?speaker i-dave)
 (?board {instance i-board1})
 (?marker {instance-of c-marker}))

((?speaker i-dave)
 (?board {instance i-board1})
 (?marker {instance i-marker3}))
```

Figure 3.4: A process segment and the development of the binding set as it executes

- **Memory and Bindings**: These actions (`assert`, `retract`, `bind`, `bind-process-id` and `unbind`) change the Classroom's internal memory state. The `assert` and `retract` actions add and remove propositions from the Classroom's memory. As is discussed in the Memory section, the Classroom's memory is not simply the set of propositions that have been asserted. But, from the perspective of a process, the memory appears to be. The `bind`, `bind-process-id` and `unbind` actions alter the process' bindings set. The `bind` action performs a memory query for the sole purpose of binding variables. The query is assumed to succeed. If it fails, the process itself fails as well; something has gone wrong. The `bind-process-id` action binds the given variable to the (unique) identifier for the given process, plan or skill instance. The `unbind` action removes the bindings in the bindings set corresponding to the given variables.

- **Skills and Pipelines**: These actions (`enable`, `connect`, `set-param` and `require`)

```
...
(steps
 (_1 (achieve _a (holding ?me ?marker)
     (assert (video-segment i-simpsons-2f08 100 200))
     (bind (video-segment i-simpsons-2f08 ?start ?end))
     (bind-process-id ?achieve-holding _a)
     (connect _skill-1 output1 _skill-2 input2)
     (do _circle (play-video-segment i-simpsons-2f08))
     (enable _timer (countdown-timer 30))
     (fork _2)
     (require speaker-events)
     (retract (video-segment i-simpsons-2f08 ? ?))
     (set-param _timer warning-time 10)
     (signal :almost-done)
     (track _brightness i-overall-brightness)
     (unbind ?start ?end)
     ...)
...
```

Figure 3.5: Examples of how each action can be used

provide an interface to the Skill Manager, allowing a process to control what skills are active, how they are connected, and how they are configured. The `enable` action instructs the Skill Manager to activate a skill based on the given invocation. Skills (as well as pipelines sub-processes) are automatically disabled when the step has completed; there is no need for a "disable" action. The `connect` action instructs the Skill Manager how to connect two skill modules together: which output to connect to which input. The `set-param` action sets one of the skills runtime parameter values. The `require` action tells the Skill Manager to make sure that the named pipeline is running (enabling the skills only if necessary). A pipeline is shared between all the processes that require it. To facilitate parameter setting and connecting skills to the pipeline, skill modules in the pipeline may be given global names.

- **Processes and Plans**: These actions (`do`, `fork` and `achieve`) provide a means

of controlling the higher-level operation of the Classroom. The `do` action instructs the Process Manager to execute a new process as a sub-process of this one, using the given process invocation. This process will be automatically disabled when the step has completed. The `fork` action is similar to the `do` action, but instead of a new process, it spawns a new sub-process within the current one, jumping to the given step. The forked process uses a copy of the process' bindings set. In the Classroom, the `fork` action is used sparingly, generally in circumstances where we want one action to continue while alternating between two or more steps. The `achieve` action indicates that the main actor should pursue a plan to achieve the given goal. This action will be discussed in detail in the next chapter.

- **Monitors**: The `track` action specifies a measurable quantity whose behavior is important. Based on the particular quantity that needs to be observed, the Classroom will activate either a skill or a process that will report (as events) what the quantity does.

- **Events**: The `signal` action reports an event that has just occurred. All events (those reported by processes as well as those reported by skills) are global; each event is reported to the Process Manager, which checks whether any of the active processes are waiting for it (see Section 3.4.1). Generally however, a process is waiting for events produced by its subprocesses; the lower-level events are then interpreted in the context of the higher-level process. In the running of a process, there are several events that are automatically signaled: the starting and stopping of the process and the starting of each of its steps. These automatic events are particularly useful in synchronizing two or more

processes that are part of a plan (see Section 4.3.2).

Every step and many of the actions (in particular, those that activate skills, processes and plans) are given a label with which they can be referred to within the process and also, by the process that activated it. These labels are used in the `bind-process-id` action to specify which process or skill to bind to the variable. They are also used extensively in the event descriptions discussed in the next section. The labels can be chained together, providing a downward path to the desired process or skill. Labels all have an "_" prefix to differentiate them from other identifiers.

3.1.5 Process events

Once all of the actions have been taken in a process step, the Process Manager will wait for the occurrence of one of the events it expects. When such an event happens, the Process Manager takes the actions associated with that event; the (required) final action either tells what process step is next or terminates the process in success or failure. Event handlers pair a description of an event with a set of actions to take in response. Depending on the class of event handler, either the event description or the actions may be implicit. For example, in an ensure clause, if the constraint does not hold, the resultant action is process failure. There are four classes of event handlers that the Process Manager uses:

- **Wait-for Clauses**: A `wait-for` event handler specifies a process or skill event that the step is waiting for. This can be a simple event (such as `:done` or `:fail`) or an event with arguments (such as `(:speaker-at i_podium)`). For a simple event, it is often necessary to specify the source of the event (e.g. to specify which process failed). An event description can identify the particular process

by giving the chain of labels that lead to it. So, the event description `(_go _1 :start)` describes the event of the first step of the sub-process labeled `_go` starting. In addition to signal events, a wait-for event handler specifies what to do when a memory proposition becomes true or false.

- **Ensure Clauses**: An `ensure` event handler expresses the required behavior of a measured quantity; if the measured quantity does not behave properly, the process will terminate in failure. These conditions are tied to a `track` action that is running as a part of the process step. Among the behaviors that an ensure condition can guarantee are that the quantity is: always increasing, always non-decreasing, or remaining constant. In addition, it is possible to specify tolerances to deal with sensor noise.

- **Ignore Clauses**: An `ignore` event handler specifies an event that may occur, but that is unimportant to the running of the process. For example, while standing and lecturing, a speaker may gesture with his hands. A simple process representing his activity may not consider the gestures important in understanding what he is doing; it will ignore these gestures. The ignore events are important in the plan recognition process. Recall that, in the plan recognition process, an unexpected speaker event will result in a candidate process being rejected. The ignore event essentially tells the plan recognizer that this event should not result in the rejection of the process.

- **When-done Clauses**: A `when-done` event handler specifies what should happen when a step is done. It is used when there is not an event that can be used to determine that the step is over. For example, when a person is standing and

```
(define-process (verbose-pickup-trash)
  (main-actor (robot ?robot))
  (steps
   (_1 (do _go (go-to-trash))
   (wait-for (_go _1 _align :close)
             (assert (speak "I am close to the trash"))
             :continue)
      (wait-for (_go :done)
             (assert (speak "I am aligned with the trash"))
             _2)
    ...))

(define-process (go-to-trash)
  (main-actor (robot ?robot))
  (roles (trash ?trash))
  (steps
   (_1 (bind (find-nearest-trash ?trash))
       (enable _align (align-with-small-object ?trash))
   ...)))
```

Figure 3.6: Two process definitions where the top process calls the second one and refers to one of its sub-actions

> lecturing, the way you can tell that he is done is that he goes and does something else. You cannot expect him to state, "now I am done lecturing" or to perform some gesture to indicate that he has completed this piece of lecturing. The implicit event referred to by a when-done event is the actor deciding that he is done–this is an event that cannot be sensed using physical sensors.

Figure 3.6 shows how actions and events fit together in the steps of two processes. The first process' first step invokes the second process, meaning that the robot will execute the second process as a part of executing the first. Of particular note is the highlighted line of the first process definition. This wait-for clause waits for an action that will be produced by the second process (a sub-process of the first). The chain of labels ("`_go _1 _align`") specifies that the first process cares about the event of the `_align` action in the `_1` step in the second process announcing that it is nearly

aligned.

The events given in wait-for and when-done clauses are actually descriptions of events–to be matched with the actual instances of events (when the event actually occurs). As a result, in the Classroom's implementation, the event descriptions are transformed into description structures that encapsulate all the important information about the event, including its name, arguments and source. When an event occurs, it is transformed into an instance structure that can then be matched to see if it satisfies the event description.

3.2 Following along with processes

In this section, we look at how the process representation outlined in the previous section is interpreted by the Process Manager. We will step through the execution of two processes and, in the second, consider the process from the perspective of the speaker (an executor) and the perspective of the Intelligent Classroom (an observer). Then, given the basic algorithms shown in the examples, we provide a detailed description of the Process Manager and its algorithms.

3.2.1 "Play a video segment"

Figure 3.7 shows the process representation used by the Classroom to play a video segment. The process consists of two steps: the first cues up the tape and the second sets all the A/V components up and plays the segment. The actions for setting the video and audio sources back to their original values are placed in the clean-up clause so that, if the process fails, it will place the Classroom back in its earlier configuration. The set-up clause is used to bind the variables that hold the original video and audio

```
(define-process (play-video-segment ?segment)
  (main-actor (classroom ?classroom))
  (roles (video-segment ?segment)
  (set-up                                          ((?classroom i-classroom)
   (bind (video-source ?old-video-source))          (?segment i-simpsons-2f08-2))
   (bind (audio-source ?old-audio-source))
   (bind (video-segment-ends ?segment
                             ?start ?end)))
  (steps                                           ((?end 320475)
   (_1 (enable _cue (cue-video-tape ?start))        (?start 320175)
       (wait-for (_cue :done) _2))                  (?old-audio-source i-cd-player)
   (_2 (assert (video-source i-vcr))                (?old-video-source
       (assert (audio-source i-vcr))                   i-mac-display)
       (enable _play                                (?classroom i-classroom)
               (play-video-segment ?start ?end))    (?segment i-simpsons-2f08-2))
       (wait-for (_play :done) :done)))
  (clean-up
   (assert (video-source ?old-video-source))
   (assert (audio-source ?old-audio-source))))
```

Figure 3.7: The process representation used by the Classroom in playing a video segment (left) and the bindings set after invoking the process and after completing the set-up (right)

sources. This guarantees that the variables will be bound regardless of what point the process terminates. (Note: this is not absolutely necessary because if there are unbound variables in an action invocation, the action simply fails.)

When the process is first invoked, the Process Manager binds variables for the process argument and the roles resulting in the bindings set shown in Figure 3.7 (on the right). The desired video segment is specified with the identifier `i-simpsons--2f08-2`, which refers to an instance structure that holds all the information needed to cue the videotape. Then, the single action in the first step is taken, starting up a skill that controls the VCR as it cues up the tape and reports when it is done. The single wait-for clause waits for the event that the skill reports and advances the process to the second step.

The second step consists of three actions. The first two actions set the Classroom's

```
(define-process (move-to-board-and-write)
  (main-actor
   (person ?speaker))
  (roles
   (markerboard ?board)
   (marker ?marker))
  (set-up
   (ignore speaker-speaking-events)
   (require speaker-motion-events))
  (steps
   (_go (do _move (move-to ?board))
        (wait-for (_move :done) _do))
   (_do (do _pickup (pick-up-from ?marker ?board))
        (wait-for (holding ?speaker ?marker) _3))
   (_3 (require speaker-writing-events)
       (fork _4)
       (when-done :done))
   (_4 (wait-for (:speaker-started-stroke ? ?) _5))
   (_5 (wait-for (:speaker-ended-stroke ? ?) _4))
  (time-constraints
   (process-duration (range 30 3000) (expected 300))
   (step-duration _go (range 0 30) (expected 5))
   (step-duration _do (range 0 1) (expected 0))
   (step-duration _3 (range 0 5))
   (step-duration _4 (range 5 3000) (expected 300))))
```

Figure 3.8: A process definition for the speaker going to the marker board and writing

audio and video inputs to the outputs of the VCR and the third action starts up a skill the controls the VCR as it plays the video segment. The single wait-for clause waits for this third action to report that the segment has been played. When this event occurs, the process is declared done and the two clean-up actions are taken, restoring the original audio and video inputs.

3.2.2 "Go to the board and write" 1st Person

Figure 3.8 shows the Classroom's representation of the process for the speaker walking over to a marker board and writing something. In this illustration, we will consider how a lecturing robot might execute the process. I do not propose that this is how a human speaker would think about going to the board and writing, but rather I

suggest that this is a useful way to go about observing a human speaker as he goes to the board and writes. The purpose of this example is to show a relationship between doing something and observing someone else doing the same thing. Because this process representation is actually used by the Classroom for observing a speaker, the actual details of what is said and what is written are abstracted away–the Classroom only cares that the speaker may say something and will write something.

When the process is first invoked, the Process Manager binds the variables for the roles. This binds the variable `?marker-board` to a description of a marker board and the variable `?marker` to a description of a marker. The `ignores` actions in the `set-up` clause indicate that the speaker is free to speak and gesture throughout the execution of the process. The `require` action in the `set-up` clause indicates that the speaker's location in the Classroom is important.

Once all the actions in the `set-up` clause have been taken, the speaker should move on to the first step of the process: moving to the board. The sole action in the step invokes a process that will move the speaker to one of the marker boards in the Classroom. When the speaker arrives at one of the marker boards, this will be reported and will satisfy the event description in the wait-for clause in the step. The event instance will hold which board the speaker arrived at and so, in matching the instance to the description, the binding for the variable `?marker-board` will be refined to refer to the particular board that the speaker will be writing on. Also, the event will advance the process to the second step.

The second step in the process instructs the speaker to pick up a marker from the marker-board tray. Similar to the first step, this step invokes a sub-process to handle the actual picking up, waits for the sub-process to complete, and refines the binding

Figure 3.9: Images corresponding to the key actions in the "go to the board and write" process.

for the `?marker` variable.

The third, fourth and fifth steps describe how to mime the writing process. In this process, writing consists of a series of pen strokes: the speaker positions the pen on the board (the fourth step) and then slowly moves it across the board (the fifth step). (This representation of the writing process is useful in the Classroom's task of keeping track of what has been written on the board.) The `require` action in the third step indicates that the speaker's writing actions are important in this step and its sub-processes. The `fork` action branches to the two steps that handle the writing; since these are forked to, the pipeline monitoring writing events will continue to run while these steps execute. When the speaker decides that he is done writing, the process terminates, as specified by the `when-done` clause in the third step.

3.2.3 "Go to the board and write" 3rd Person

In this example, we again look at the process definition given in Figure 3.8, but instead look at how it is actually used in the Classroom: to follow along with the speaker as he writes on the board. So, while the previous example was rather artificial, this example demonstrates the essential algorithms that the Classroom uses in monitoring

the progress of the speaker in a process. As we will see in the next chapter, these algorithms are also crucial to the plan recognition process. Figure 3.9 shows camera images corresponding to the key events in the execution of this process.

Initially, the speaker is standing and lecturing in the podium region of the Classroom and (magically[1]) the Classroom determines that speaker is going to move over to the marker-board and write. As in the previous example, the first thing that happens is that the Process Manager binds the variables for the roles. But, this time the `require` action in the `set-up` clause takes on a more practical significance: since the speaker's location is important to the progress of the process, the Classroom needs to make sure that the set of skills to monitor it is active. These skills produce events such as `(:speaker-at i-podium)` and `(:speaker-left i-screen)` and `:speaker--moved`. Because this pipeline is activated in the `set-up` clause, it will remain active throughout the execution of the process.

Next, the Process Manager will execute the first step of the process: moving to a marker-board. The step itself spawns a sub-process that deals with lower-level details of watching the speaker move. This sub-process (shown in Figure 3.10) will make sure that the speaker is making progress towards one of the boards. If the speaker moves away or stops before reaching one of the boards, the sub-process will fail–the speaker must not be going to the board to write. The step is only waiting for the event that indicates that the speaker has arrived at one of the marker-boards. This event will result in the `?marker-board` variable being bound to the appropriate marker-board.

The second step is much like the first: it spawns a sub-process that observes

[1]In this example, we are not concerned with how the speaker and Intelligent Classroom simultaneously determine that the next portion of the lecture will involve writing on the marker board. In practice, this could be specified in a presentation script (see Section 3.3) or the Classroom could infer that the speaker was going to the board to write, using plan recognition (see Chapter 4).

```
(define-process (move-to ?location)
  (main-actor
   (person ?speaker))
  (roles
   (floor-location ?location))
  (set-up
   (require speaker-motion-events))
  (steps
   (_1 (wait-for (:speaker-approached ?location) _2)
       (wait-for (:speaker-entered ?location) _3)
       (wait-for (:speaker-stopped) :fail))
   (_2 (wait-for (:speaker-entered ?location) _3))
   (_3 (wait-for (:speaker-stopped) :done)
       (wait-for (:speaker-left ?location) :fail)))
  (time-constraints
   (process-duration (range 1 15) (expected 5))))
```

Figure 3.10: A process definition for the speaker moving to a particular floor region

the speaker as he picks up a marker from the marker-board tray. The Classroom remembers where on the tray the speaker has placed the various markers and erasers, and so, based on where he reaches, it can determine what he picked up. When the speaker has picked up a marker, this is reported as an event that results in the `?marker` variable being bound to the appropriate marker.

The third, fourth and fifth steps observe the writing process. The third step enables a pipeline that observes and reports writing events. This pipeline remains active while the fourth and fifth steps are running. The Classroom is not interested in what the speaker writes (i.e. it does not try to read his writing or comprehend his figures), but is instead interested in the particular regions of the board that the speaker has written on. This proves important in filming the presentation; if the speaker points at something on the board and the Classroom knows the boundary of what he is referring to it can frame only the relevant portion of the board. Similarly, in presenting reference materials to students after a lecture, the Classroom can show the

development of a figure that was repeatedly updated by the speaker. To accomplish this way of understanding what the speaker is writing purely spatially and temporally, it represents writing as a series of marker strokes. The fourth step looks for the start of such a stroke (i.e. that the speaker is holding the marker nearly stationary over the board) and the fifth step follows the motion of the marker across the board and notices when the stroke is complete (i.e. that the speaker move the pen rapidly to a new location). The `when-done` clause in the third step indicates that the way we can tell that the speaker is done writing is when he starts doing something else. So, the Process Manager will switch between the fourth and fifth steps until the speaker does something that indicates he is no longer writing at the board (e.g. moving away from the board).

3.3 Presentation scripts

Presentation scripts are processes that represent entire presentations. In the simplest case, they aid the Intelligent Classroom in cooperating with the speaker by always telling the Classroom what will happen next. In this case, rather than relying on plan recognition and risking making mistakes, the Classroom only needs to recognize when the speaker has gone from one step to another–a much simpler task. By reducing the potential ambiguity of the speaker's actions, a presentation script can help the Classroom react more quickly and accurately. In a typical lecture that uses a presentation script, the presentation script will give a rough outline of the key events in the lecture and plan recognition will be used to handle the details.

In complicated lectures, presentation scripts aid the Classroom in cooperating with the speaker by telling it how to react to particular events in particular situations,

```
(define-process (give-anatomy-lecture)
  (main-actor (speaker ?speaker))
  (steps
   ...
   (_6 (require speaker-touching-events)
       (wait-for (:speaker-touched-wall ?x1 ?y1) _7))
   (_7 (require speaker-touching-events)
       (wait-for (:speaker-touched-wall ?x2 ?y2) _8))
   (_8 (bind (bounding-box ?x1 ?y1 ?x2 ?y2 ?box))
       (set-param _*camera* rectangle ?box)
       (set-param _*camera* mode zoom-on-rectangle)
       (enable _timer (countdown-timer 15))
       (wait-for (_timer :done) _9))
   ...))
```

Figure 3.11: A presentation script fragment detailing the touching of the two ends of a bone to specify where to zoom the presentation camera

Figure 3.12: The speaker points at both ends of the skeleton's ulna and then the presentation camera zooms in on the bone.

allowing interaction that would not be possible using the Classroom's default cooperative behaviors. For example, in an anatomy lecture, the speaker might want the Classroom to zoom in on the appropriate bones in a skeleton. Without a knowledge of skeletal structure and without the ability to see the ends of bones, the Classroom cannot do a good job of showing the skeleton's parts. In this situation, a presentation script can provide a way for the speaker to show the Classroom what region to show in the video of the presentation. Figure 3.11 shows a fragment of a presentation script

that allows the speaker to specify a bone to zoom in on by touching both of its ends. After the speaker touches both ends, the camera zooms in on the specified region for a moment as shown in Figure 3.12.

A presentation script is basically a process that the Process Manager refuses to fail. When the speaker deviates from the script, the Process Manager will suspend it, waiting for the speaker take an action that indicates he is resuming the script. That is, the Process Manager waits for events in the current and next steps in the presentation script. The idea here is that, in the course of a presentation, the speaker may temporarily go off on a tangent, but he will return to the planned presentation eventually.

It is our intention that presentation scripts will not be required in typical presentations. That is, we hope that the plan recognition will detect and react to the actions commonly used in presentations. Presentation scripts ought to be used to aid the Classroom in situations where the structure of the presentation is well known or where the presentation contains actions that the Classroom will not know how to respond to.

3.4 Implementation

The examples given in the previous sections provide an intuition for what the Intelligent Classroom must do in following along with processes. In this section, we look in greater detail at the structures and algorithms that make it work. The Process Manager keeps track of all the active processes: their hierarchical structure, their progress and what events they are waiting for. In this chapter we will ignore (for a moment) a few ways that the algorithms are changed for performing plan recognition.

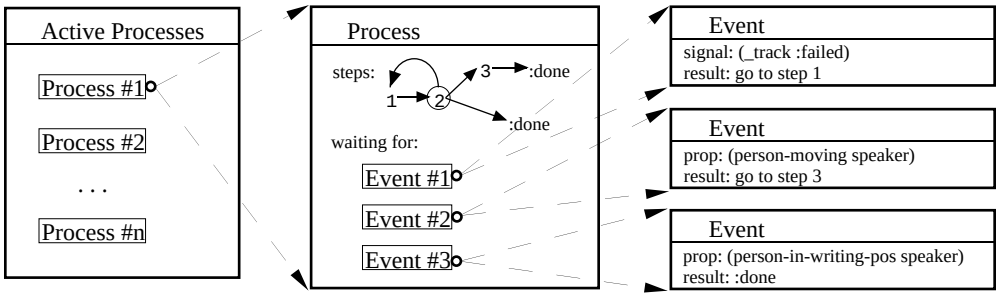


Figure 3.13: Important structures in the Process Manager

3.4.1 The Process Manager

Figure 3.13 shows a block diagram of the Process Manager. On the left are the process set (the set of all the currently active processes) and a simplified representation of the components of a process instance. On the right are the set of all the events that the Process Manager currently is concerned with and a simplified representation of the components of a wait-for event instance.

Each process instance has a parent process (i.e. the process that spawned it), a set of the child processes[2] that it has spawned, including a special child process for its current step, its set of bindings, and a set of event handlers for all of the events that it is waiting for. A process is instantiated with a parent process (possibly the unique "root" process), a main actor (usually either the speaker or the Classroom), and a process invocation (the name of the process and values for all of its arguments). When a process is instantiated, the Process Manager:

Builds a process instance structure for the process and ties the new structure to

[2]In this section, for lack of a better word, I use "sub-process" to refer to any continuous action that has been spawned by a process instance. These sub-processes may be proper processes, skills, or pipelines. Perhaps a reasonable definition of "sub-process" for this section would be "continuous actions that will need to be terminated when this process stops". I often refer to non-process continuous actions as "special" sub-processes.

its parent, filling the appropriate parent and child slots. This process is added to the process set.

Tries to bind the main actor and argument variables using the given main actor and process arguments. If this fails, the process is immediately failed, reporting its failure as an event. If the binding succeeds, the resulting bindings set is added to the process instance structure. At this time, the Process Manager also adds bindings for all the process roles.

Executes any actions in the process' `set-up` clause. If any of these actions immediately fail, the process is failed as well. For those actions that spawn non-process continuous actions (e.g. skills), a special sub-process structure is added to this process' set of children, keeping track of it.

Starts the first step. The Process Manager creates a special "step" sub-process for holding the step and its processes and links it to the main process. Then, all of the step actions are executed, using this step sub-process as their parent. At the same time, the Process Manager deals with the event handlers. For each of these, a description of the event is constructed, linked back to this process and the event description is added to the list of events that the Process Manager is currently concerned with. Time constraints are transformed into event handlers as well; if a process step is constrained to take less than 30 seconds, the Process Manager adds an event handler that essentially says, "if this step has not completed by the time 30 seconds have elapsed, the time constraint has been violated and so we should fail."

Once the process is instantiated, the Process Manager simply waits for one of the events that this process cares about to occur. When an event occurs, it is reported to the Process Manager as either an atomic identifier (e.g. `:done` or `:speaker-moved`)

or an identifier with appropriate arguments (e.g. `(:speaker-stopped-at i-board-1)`). The Process Manager also cares about which process reported the event. While the event source is less important for events about what the speaker is doing, it is critically important in reporting the starting or stopping of sub-processes. When an event is reported to the Process Manager, it:

Builds an event instance structure for the event. (Recall that an instance is a description tied to an identifier.) The event description is of the event class associated with the event's identifier and the description's slots will be filled with the event arguments and the process that reported the event.

Checks the event instance against all of the event descriptions in its event set. (The algorithm for checking whether one description satisfies another is given in the Descriptions section in this chapter.) The Process Manager responds to every match, rather than just the first, because there is often more than one process waiting for a particular event.

(For each match) tells the process associated with the matched description to deal with the event. First, if appropriate, the process' bindings set is updated based on the event arguments. Then the Process Manager executes each of the event handler actions (as in step 4 of the process instantiation algorithm) using the step sub-process as the parent. Finally, the final action is taken. If the result action is `:continue`, the Process Manager just goes back to waiting for events. If the result action is a step identifier, the process is advanced to the step associated with that identifier. If the result action is `:done` or `:fail`, the whole process is stopped and the result action reported as an event for this process.

When a process is advanced to a different step, the Process Manager first stops

all of the current step's sub-processes and then starts the new step, as detailed in step 4 of the first algorithm. In halting the current step, the Process Manager looks at each of its sub-processes, in the reverse of the order they were started, and takes an appropriate action to stop them. A required pipeline is released; an enabled skill is disabled; a running process is halted. However, the Process Manager does not attempt to undo the results of any non-process-based actions. So, a memory assertion or variable binding is not retracted. In addition, the events associated with the step's event handlers are removed from the list of events that the Process Manager is currently concerned with.

When a process is halted, as in when the result action is `:done` or `:fail`, the Process Manager first halts the current step, stopping its sub-processes, then stops all the sub-process started in the process' `set-up` clause, and finally performs all of the actions in its `clean-up` clause. Having effectively halted the process, the Process Manager then reports the process' result action as an event from the process. All processes implicitly wait for `:fail` events from their sub-processes, with the default result of failing when a child process fails.

3.4.2 The Memory System

The Process Manager views the internal state of the Intelligent Classroom as a huge database of facts, expressed in first-order predicate calculus. The Memory System is responsible for making this illusion work, for, while many facets of the Classroom's state are easily represent as simple facts, many others can be much more efficiently represented in other ways. In the Memory System, we attempt to make the most of the simplicity of predicate calculus and also to use the most natural underlying representations. For instance, in the Classroom, it is often useful to know what

speaking area (e.g. the podium, a marker-board or the display screen) the speaker is closest to. It would be possible to maintain a database of facts about the various spatial relationships that hold in the Classroom at any given moment, updating the database every time anything moved. But, it is much more efficient to just store the locations of the objects and then infer the needed information whenever it is needed.

The Memory System provides an interface through which any component (e.g. the Process Manager) can access much of the internal state of the Classroom through simple queries, assertions and retractions of propositions. A memory proposition takes the form: (<proposition-name> <arg>*). Each type of proposition (all propositions with a given name form a type) is registered with the Memory System, telling the Memory System what to do when a query or assertion is made. So, when a query is made, the Memory System calls the appropriate query function in its table of proposition types, passing the queried proposition and (when appropriate) the bindings set as arguments. This function returns the set of all bindings sets that satisfy the query, as a lazy list[3]. Similarly, when an assertion or retraction is made, the Memory System calls the appropriate assert or retract function, resulting in the appropriate change in the Classroom's internal state.

In addition to simple propositions, a query may also be composed of various Boolean operators, such as `and`, `or` and `not`. These operators work with the bindings sets returned by the memory queries that are their arguments. Figure 3.14 shows a rather simple query designed to select a blue marker. The first proposition in the

[3]A lazy list in Lisp is a data structure that, through delayed execution, allows programmers to efficiently deal with potentially troublesomely large lists. The basic idea is that items in the list are computed as they are accessed, rather than all at once. So, if only a few items of a list are expected to be needed, a lazy list can be significantly more efficient. For example, a lazy list can represent the list of all positive integers; whenever the program tries to move past the end of the computed list, another element is automatically added.

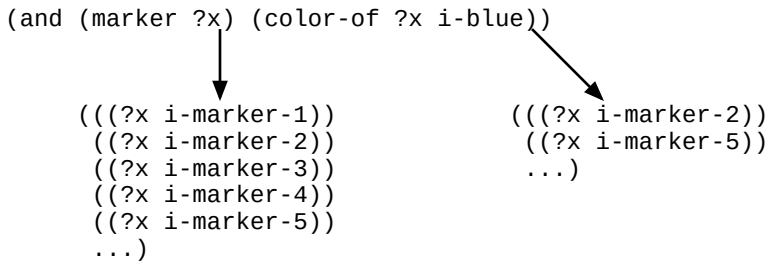


Figure 3.14: A compound query and the sets of binding sets produced in performing the query. Markers #1, #3 and #4 are not blue.

query returns bindings sets for each marker that the Classroom knows about. Then, each bindings set is tried with the second proposition to see if the marker is blue. The result of this query will be a lazy list of bindings sets with the variable `?x` bound to each blue marker in turn.

From the perspective of the Memory System, the Classroom is seen as a set of about ten components, each providing a proposition-based interface to control and query its state. For example, some of the Classroom's simpler hardware components (e.g. the video source switcher) provide a proposition-based interface so that they can be controlled directly within processes, through memory assertions and retractions.

But, due to how processes often wait on memory propositions, rather than just on events, the Memory System also needs to provide some functionality to make it so that the Process Manager does not have to repeatedly poll (i.e. "busy wait") the Memory System regarding the memory propositions it cares about. So, similarly to how components may register a proposition type with the Memory System, components may also register that they are interested a particular proposition type–whenever an assertion of that type is made, the component will be informed via the "snoop" function it has registered. In addition to improving efficiency, the snoop functions also

provide an invisible way of monitoring the internal state of the Classroom. In the future, we hope to automate the analysis of snooped data, allowing the Classroom to learn some of the tendencies of particular speakers who lecture there.

One difficulty with using the snoop functions for the propositions that the Process Manager is waiting for is that, for some propositions, it is somehow difficult to make the appropriate assertions. For example, in the Process Manager, one process may be waiting for another process to start. Ideally, the Process Manager would use its own Memory System interface to invoke processes, but, due to the complexity of the process data structures, this is not feasible. In such cases, the Memory System allows a component to register a proposition type slightly differently: rather than allowing other components to assert changes (e.g. to invoke processes) to it, the component promises to report when it changes itself (e.g. it invokes a process). To other components, this ends up looking like the assertion that they were waiting for.

Another difficulty with snoop functions occurs in components like the spatial relationships component discussed at the beginning of this section. As mentioned before, it is very inefficient to compute all of the spatial relationships whenever anything moves. So, in this case, we are forced to choose between repeatedly computing the spatial relationships and busy waiting. In the Classroom, we have found that busy waiting is more efficient with our current hardware setup. The important point is that the Memory System and the event-handling portion of the Process Manager support both methods. Each proposition is marked with the appropriate method, and this method is then used in the Process Manager.

3.5 Open issues

The Process Manager, as described in this chapter, does a reasonable job at following along with the rather straightforward processes currently used in the Intelligent Classroom. But, the algorithms are fragile in the face of sensor failure (i.e. sensor noise resulting in the Classroom missing or hallucinating events) and process failure (i.e. processes and process steps not producing their intended results). These issues are not devastating in the relatively forgiving Classroom environment. Many of the mistakes the Classroom may make will go unnoticed and the speaker can easily remedy most of the rest, by issuing a voice command or just ignoring the mistake and going on. But, if the Process Manager is applied to more dynamic and less forgiving domains, such as mobile robotics, these issues will become crucial. In this section, we briefly look at these two issues and suggest future research directions that may help deal with them.

3.5.1 Dealing with sensory noise

One of the greatest weaknesses of the Process Manager is its vulnerability to sensory noise. If the Classroom's sensors miss a crucial event, the process waiting for that event will not advance and it is likely that the process will eventually fail as a result. Similarly, if the Classroom' sensors hallucinate an event, some processes waiting for that event may be mistakenly advanced, often leading to their untimely termination. In the next chapter we will show that, due to the way plan recognition works in the Classroom, this problem does not prove critical with the short and flexible plans occurring in this domain.

However, if the Classroom is following along with a presentation script, it is very

important that this particular process does not fail; it provides valuable information about how to interpret the speaker's actions. For this reason, the Process Manager treats presentation scripts slightly differently than a regular process. If the speaker seems to take an action that conflicts with the presentation script predicts, the Process Manager suspends the presentation script, waiting for the speaker to take an action that indicates he has resumed his planned presentation. Essentially this entails waiting for events from the current step in the presentation scripts and also events from all the steps to could immediately follow the current one.

This technique of suspending a process and then waiting for the speaker to indicate that he has resumed it seems to work well for presentation scripts and in cases where the vision system hallucinates that the speaker has moved. But, in general, this is not a satisfactory solution. It results in a Classroom that is overly tenacious, refusing to give up on a failed process, even in the face of overwhelming evidence.

In Jabberwocky (discussed in Chapter 5), we have tried introducing probability as a technique to deal with extremely noisy voice recognition results. The basic idea is that a set of slides are treated as a presentation script and the words and phrases the Classroom hears are used as evidence for which slide the speaker is lecturing from. The evidence is accumulated in terms of probabilities based on word and phrase frequencies. The speech recognition results tend to be very poor in a lecture setting; error rates of sixty to seventy percent are typical. In this extremely noisy situation, probability seems like the only obvious solution. But the other sensors in the Classroom are much more reliable and it is not clear how we should compute the hundreds of conditional probabilities such an approach would require.

One approach that would probably substantially improve the Classroom's ability

to deal with sensor noise without changing the operation of the Process Manager would be to alter the processes themselves to deal with the noise. To accomplish this, the Classroom would observe numerous presentations and learn where, how and when events are missed, discovering the particular situations where the sensors are likely to either miss events or hallucinate them. Then, the Classroom can add event wait-for clauses to accommodate the events produced by sensor noise.

Another approach that seems promising is to use a much looser notion of probability. Rather than computing precise probabilities, the Classroom would have a notion of its confidence that things are going well with a process. Intuitively, when the Classroom detects events that are not consistent with the progress of a process, the Classroom's confidence in that process will fall. If this confidence falls far enough, the process is rejected. The Classroom would probably work quite well with three or four levels of confidence, and rather than needing to come up with hundreds of probabilities, we could just enumerate some common sensory noise situations and decide how devastating they are to our confidence.

3.5.2 Dealing with failure

The Classroom's Process Manager was originally conceived as a revised version of the RAP system [29], enhanced to provide the plan recognition capability needed to cooperate with other agents. The RAP system is built around the notion that if one approach does not work, another method is selected which will accomplish your goal or at least get you closer. Each method for accomplishing a goal has an associated context in which it is applicable and the RAP system keeps trying methods until it succeeds or decides that the situation is hopeless.

The Classroom currently does not deal with failures of the sort that the RAP

system deals with so naturally. The processes that the Classroom executes tend to only have one method. If that method fails, there is probably no other way to do it. But, even when the Classroom's processes do exhibit the multiple method property indicating that a RAP-like approach will work well, it is not clear that a straightforward adaptation of the RAP method chooser is the correct way of making the Classroom deal better with failure. The problem is that the Classroom needs to understand both its own failures and those of the speaker. It is unreasonable to assume that the speaker will use the same method selection algorithm as the Classroom, but it is also apparent that the speaker will, in some situations, do something analogous. If the Classroom is able to reason about the speaker's failures, it will be able to understand what the speaker is doing in the face of failures and, in some situations where the speaker prematurely gives up, could even suggest possible solutions to him.

Chapter 4

Plan recognition and cooperation

In the previous chapter, we looked at how the Intelligent Classroom represents continuous activities (i.e. processes) and how it follows along the processes going on within it. This would be sufficient for an implementation of the Classroom where the Classroom knows precisely what the speaker is going to do and just needs to stay synchronized with him. Unfortunately, this never happens. Since the speaker will sometimes take actions that the Classroom is not explicitly expecting, the Classroom must be able to infer the speaker's plans from his actions and then figure out what actions to take to cooperate.

So, in this chapter, we look at the plan representation used in the Classroom and how the Process Manager is used to recognize the speaker's plans and take cooperative action. In both plan recognition and cooperation, the Process Manager does most of the work; the algorithms outlined in this chapter simply add event handlers that produce the desired behavior.

Recall from the Introduction that the Classroom's plan recognition differs significantly from most plan recognition research: the Classroom is much more concerned with *what* the speaker is doing than *why* he is doing it. This is because the Classroom

directly uses the inferred plans to produce cooperative behavior, rather than reasoning about the speaker's goals in order to select an appropriate course of action. Also, the Classroom performs plan recognition in a robotic domain, using the readings from physical sensors, rather than interpreting natural language or other high-level action representations. For both of these reasons, the plan recognition techniques described in this dissertation do not do the goal-level reasoning that is described in much of the plan recognition literature. Given the Classroom's task and domain constraints, such reasoning is unnecessary and would be exceedingly difficult, if not impossible, to employ.

4.1 The philosophy and representation of plans

From the process-based representation of the world, the Intelligent Classroom has an understanding of what is happening; from its plan-based representation, the Classroom has an understanding of why it is happening. As defined in the Introduction, a *plan* is a set of processes (often to be executed by a number of different agents) that, when run together successfully, accomplish some goal. In the Classroom, most of the plans have the speaker executing one process while the Classroom executes one or more processes in cooperation. The plan representation has the advantage of making the presence of multiple agents explicit–everyone who has a role in pursuing the plan has a process to execute that defines their role in the plan. Informally, a plan says, "If you wish to achieve this goal, here are all the processes that should be run, and here is how they need to fit together." In a cooperative system such as the Classroom, this representation facilitates cooperation by essentially telling the system, "If you see someone executing a process from this plan, they are probably pursuing this plan

```
(define-plan (move-to-board-and-write)
  (main-actor
   (person ?speaker))
  (roles
   (intelligent-classroom ?classroom))
  (accomplishes
   (?speaker
    (do (move-to-board-and-write))))
  (processes
   (_p1 ?speaker
        (move-to-board-and-write))
   (_p2 ?classroom
        (film-move-to-board-and-write
          ?speaker)))
  (synchronization
   (starts (_p1 _go) (_p2 _go))
   (equals (_p1 _4) (_p2 _do))))
```

```
(define-process (move-to-board-and-write)
  (main-actor
   (person ?speaker))
  (roles
   (marker-board ?board)
   (marker ?marker))
  (steps
   (_go (do _move (move-to ?board))
        (track _delta (track-floor-distance ?speaker ?board))
        (wait-for (_move :done) _do)
        (ensure _delta (:decreasing)))
   (_do (do _choose (select-marker ?marker))
        (wait-for (_choose :done) _3))
   (_3 (do _pickup (pick-up-from ?marker ?marker))
       (wait-for (holding ?speaker ?chalk) _4))
   (_4 (do _write (write-on-board ?board))
       (wait-for (_write :done) :done)))
  (time-constraints
   (process-duration (range 30 3000) (expected 300))
   (step-duration _go (range 0 30) (expected 5))
   (step-duration _do (range 0 1) (expected 0))
   (step-duration _3 (range 0 5))
   (step-duration _4 (range 5 3000) (expected 300))))


(define-process (film-move-to-board-and-write ?speaker)
  (main-actor
   (intelligent-classroom ?classroom))
  (roles
   (person ?speaker))
  (steps
   (_go (do _film1 (film-moving-speaker ?speaker))
        (wait-for (_film1 :done) _do))
   (_do (do _film2 (film-writing-speaker ?speaker))
        (wait-for (_film2 :done) :done))))
```

Figure 4.1: Plan and process definitions for the speaker going to the marker board and writing

and so you should execute these other processes in order to cooperate."

Figure 4.1 shows part of the plan and process representation that the Classroom uses when the speaker walks over to the marker board and writes. The plan definition (on the left) consists of several parts, common to all plans:

- The *invocation* defines how this plan is to be invoked. If the plan is invoked as a part of a process, the invocation may specify arguments that define how the plan should proceed (e.g. where to go or what object to pick up). If the plan's presence is inferred through plan recognition, the invocation may be used indirectly to define arguments in the plan's sub-processes. As in process

invocations, the first item (`move-to-board-and-write` in the example plan) is the name given to the plan. Any other items in the invocation serve as arguments to the plan. The variable binding process in plans is the same as it is in processes.

- The *main-actor definition* describes what kind of agent may be expected to pursue this plan. In the Classroom, this definition is used to distinguish between plans that the speaker is likely to pursue and those that Classroom itself may pursue. In a domain such as the Classroom, where the roles of speaker and Classroom are so clear, this distinction may seem artificial, since every plan is exclusively for either the speaker or the Classroom. But, in a domain where the agents are more homogenous, it is easy to imagine plans where the roles are less strict.

- The *role definitions* describe the various agents and objects that play an important part in the execution of the plan. As in process definitions, each role definition describes what can fill that role and defines a variable for referring to the role filler. Role definitions are often used to describe the arguments given in the process invocation. In the example plan, the only role (besides the main actor role) is that of the Classroom.

- The *accomplishes clause* gives a set of reasons to invoke this plan (i.e. goals that it achieves). One of these reasons can take two forms: (a) goal achievement, in which the plan is used to make some memory proposition become true, and (b) action accomplishment, in which the plan is used to allow an agent to take some course of action. The first sort of reason is useful in dealing with

the sorts of goals that are naturally represented as a memory proposition (e.g. `(achieve (at ?i-speaker ?i-podium))`). However, `do` goals are used in situations where it is difficult to express the reason for pursuing a plan in terms of a memory proposition to achieve. The example plan is in service of such a reason; it would be very awkward to formulate a complicated memory proposition that would be made true if the speaker successfully moved to the board and wrote something.

- The *process definitions* describe the processes that need to be executed as a part of this plan. Each process definition specifies a label for the process (in the example plan: `_p1` and `_p2`), the actor who needs to execute the process, and the process' invocation.

- The *synchronization clauses* specify how the execution of the processes in the process definitions needs to be interleaved. Each clause consists of one of Allen's temporal relationships [2] and two continuous actions that need to be constrained in that way. The twelve temporal relationships define in what order the start and end points of two time intervals must occur. In the example plan, the first clause specifies that the first step of the first process starts at the same time as the first step of the other and runs longer. The continuous action specifications are chains of labels (as are also used in process time constraints and event specifications) that lead from the plan to the desired process, process step, or sub-process.

Together, these plan components hold all of the information needed to successfully execute the plan. Like the processes, the definitions for all the plans are stored in a

library of plans, in this case indexed by both plan name and by what they accomplish.

4.2 Plan recognition

Now, given the Intelligent Classroom's plan representation and its techniques for following along with or executing the processes that these plans are built out of, we are left with the task of figuring out what plan to execute when the speaker deviates from what the Classroom expects him to do. (Once the Classroom starts its portion of the plan, it is effectively cooperating with the speaker.) The Classroom determines that its understanding of what the speaker is doing is incorrect when the speaker takes an action that cannot be explained by the processes in the Classroom's understanding. That is, the Classroom is keeping track of the processes that the speaker is executing (usually this is one top-level process and all of its sub-processes) and it expects that at least one of these processes will be waiting for any speaker action that the sensing system reports. If the sensing system reports a speaker action that none of the processes were expecting, the Classroom infers that its processes do not accurately represent what the speaker is doing. This situation calls for plan recognition.

Recall from the Introduction that the plan recognition process consists of first recognizing that a new explanation for the speaker's activity is required, then proposing a set of possible explanations for what the speaker is actually doing, and finally eliminating all the incorrect explanations as future speaker actions contradict them. The Process Manager has to be modified slightly to deal specially with speaker events; this modification, which also facilitates candidate rejection, is detailed at the end of this section.

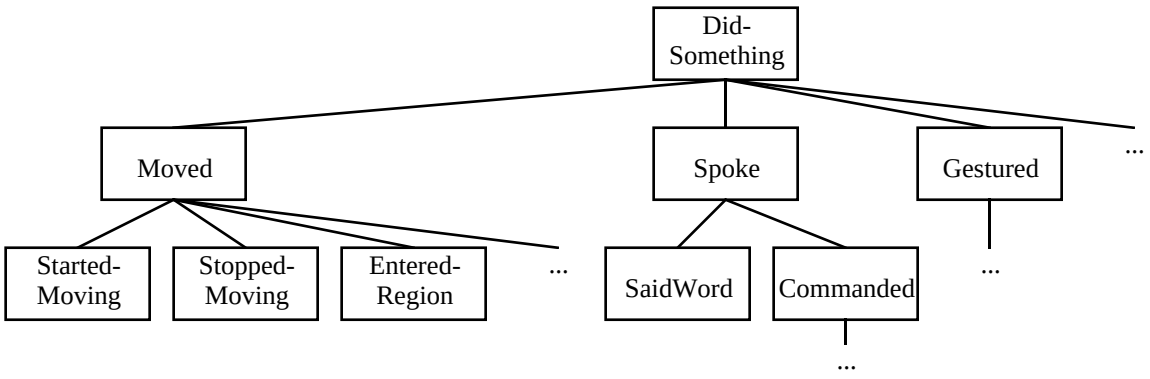


Figure 4.2: A portion of the Intelligent Classroom's speaker event hierarchy

A speaker event is explained by a process if the process has a wait-for event handler that matches the event or if it has an ignore event handler matching the event. To eliminate the need for enumerating all of the speaker events to wait for or ignore, the Classroom recognizes classes of speaker events. Figure 4.2 shows the event hierarchy used in the Classroom to represent speaker events. At the top is the most generic event indicating simply that the speaker has done something. Below that are high-level classes of events dealing with the speaker moving, speaking, gesturing and writing. At the bottom are the events corresponding to specific speaker actions. The key here is that, if a process is waiting for (or ignoring) a class of speaker events, that process explains any speaker event that is more specific.

4.2.1 Proposing candidate explanations

In the Intelligent Classroom, plan recognition is made more difficult by the fact that the Classroom does not know for certain when the process it is trying to recognize actually started. The Classroom knows which particular speaker action informed it that its understanding of what the speaker is doing was wrong, but that action does

not necessarily mark the beginning of the speaker's process. The speaker may have started his current process two or more actions previously and the Classroom failed to take notice because those actions happened to also be predicted by an existing process. As a result, in proposing candidate explanations for an unexpected speaker action, the Classroom needs to consider the last several speaker events, comparing them against the possible sequences of events predicted by potential explanations. In this chapter, I use the word "explanation" to refer to the process the speaker is executing, the phrase "representative process" to refer to the Classroom current best guess as to what process the speaker is executing, and the phrase "explanatory plan" to refer to the plan corresponding to the representative process. This distinction is necessary because of the Classroom's distinction between processes and plans.

When the Classroom observes an unexpected speaker action, it uses the last five actions in simulating the running of each possible speaker process. The idea is that, for each process, and for each of the five terminal subsequences of the actions, the Classroom sees how the process would progress if those actions occurred. Each process starts in its initial step, and if a speaker action matches one of the events it is waiting for (or ignoring), the process moves on to the specified next step; if a speaker action does not match, the process is rejected because it does not explain the what the speaker did. If the process fails or halts, the Classroom removes it from consideration. But, if not, the process has successfully explained the sequence of events and is accepted as a candidate explanation.

However, the algorithm sketched above is rather inefficient, requiring five simulations for every possible speaker process. So, in the implementation, we build an

event decision tree that is able to quickly filter out explanations that are incompatible with a sequence of actions. The root of the tree corresponds to the situation where nothing has happened–at this point all of the processes are candidates. The edges leading out of the root node correspond to possible speaker events. The events may be fully specified (if they are explicitly specified in the process) or may have unbound variables. To find which processes may be compatible with the observed speaker events, the Classroom simply follows the edges that match the events. The processes corresponding to the node (or possibly nodes due to unbound variables) are then simulated to determine whether they can be accepted as candidate explanations. They must be simulated because the event tree does not take time constraints and variable binding into consideration. In principle, this tree could be unmanageably large (i.e. the number of possible speaker actions to the fifth power). In practice, however, the branching factor after the first step will tend to be one or two, as only one or two events will be a part of any initial sequences of actions for a process.

After running this selection algorithm, the Classroom now has its set of candidate processes and further knows which step each should currently be in and what their variable bindings should be. Next, the Classroom adds these processes to the Process Manager, which will be used to continue the plan recognition, discovering what the speaker is actually doing. As a matter of efficiency, the Process Manager treats these processes specially, looking for opportunities that will allow candidate processes to share sub-processes. Also, if a candidate process is supposed to spawn a plan as a step action, the Classroom will not start its processes in the plan; it will only cooperate with the current generalization process (i.e. the Classroom's current guess about what the speaker is doing).

Finally, it is important to note that the Classroom does not consider all of the processes in its process library in selecting its candidate explanations. It only looks at processes that can be the speaker's process in one of the speaker's plans. This set of processes is also called the set of possible top-level processes because they are the speaker processes corresponding to possible explanatory plans that the speaker might be pursuing.

4.2.2 Rejecting candidate explanations

Once the Classroom has proposed a set of candidate explanations for an unexpected speaker action, it is left with the task of determining which one of these explanations corresponds to what the speaker is actually doing. It is not possible to immediately determine precisely what the correct explanation for the speaker's actions is. Early on, there is simply not enough information. But, as the speaker takes additional actions, the Classroom can discover exactly what he is doing. The Classroom uses the Process Manager to rule out the incorrect processes–a candidate explanation can be rejected for any of the reasons that would lead a process to fail.

When the Process Manager fails a candidate process (due to a time constraint violation or an event occurring with a resultant action of `:fail`) the process is not only removed from the Process Manager's set of active processes; it is removed from the set of candidate explanations. Essentially, the Classroom is waiting for `:fail` events from each of the candidate processes. When such an event occurs, the candidate explanation set is refined. When the set is reduced to a single process, that explanation is accepted as being what the speaker is actually doing.

A second way that candidate processes can be eliminated involves their ability to explain what the speaker does; when the Classroom observes a speaker action,

each candidate process must be waiting for (or ignoring) that action. For example, if one of the candidate explanations were that the speaker was going over to point out something on the projection screen, we would not expect him to stop at the marker board. The speaker's action of stopping at the board serves as evidence that he is not going over to the projection screen.

4.2.3 Changes to the Process Manager

In order to facilitate the plan recognition described above, the Process Manager requires a few minor alterations to its implementation (as it is described in Section 3.4.1). The Process Manager needs to treat speaker events differently than the other events[1] and to treat explanatory processes (those serving as candidate explanations and as the current best explanation) specially. The speaker events and explanatory processes use the basic functionality of the Process Manager, but also need the Process Manager to pay special attention to their plan recognition aspects.

The additional responsibilities of the Process Manager in service of plan recognition include making sure that every potential explanation of what the speaker is doing is consistent with what the Classroom observes the speaker doing. These explanations include both the candidate explanations and the accepted explanation of what the speaker is doing. The Process Manager stores these processes in its set of active processes and, in addition, it keeps track of the explanations separately to make sure that each explanation explains each speaker event.

So, when an event of any kind occurs, the Process Manager first checks whether it is a speaker event. If not, the Process Manager simply performs the standard event

[1] In the Intelligent Classroom, it is only the speaker's actions that require explanation (in the plan recognition sense). But, in a cooperative environment with more agents, a plan recognizer would need to concern itself with events corresponding to the actions of all the other agents.

processing as described in the previous chapter. But, if it is a speaker event, the Process Manager also keeps track of which explanations were expecting the event. An explanation is said to expect (or explain) a speaker event if the process or one of its sub-processes is waiting for or ignoring an event that matches the speaker event. Whenever the Process Manager successfully matches a speaker event with one of the events in its set of event descriptions (i.e. the descriptions of all of the events that the Process Manager is currently concerned with), it takes note of which process was waiting for the event. Then, it marks any explanations that have this process as a sub-process or, if the process itself is an explanation, it is marked. When the entire set of event descriptions has been dealt with, the Process Manager makes sure that each of its explanations has been marked as expecting the speaker event. Any explanations that are not marked are then rejected. Finally, the Process Manager adds the speaker event to its queue of recent speaker events. This is the event queue used to produce candidate explanations.

4.3 Cooperation

Figuring out what the speaker is doing plays an important role in how the Intelligent Classroom interacts with him. But, in order to cooperate, the Classroom needs to both understand the speaker's actions and then do something about it. Much of the Classroom's plan representation is designed to facilitate doing just that. Given an explanation of what the speaker is doing (in the form of a top-level process), the Classroom selects the plan in its plan library that is appropriate for cooperating with it. (For each possible speaker process there is exactly one plan for cooperating with it.) The Classroom executes its processes in this explanatory plan and makes sure

that its actions are synchronized with those of the speaker.

The Classroom's notion of cooperation is not one of interaction between peers; the Classroom is in every way an assistant to the speaker. That is, the Classroom is always looking for ways of helping the speaker and never solicits help from him. This is important to understand in looking at how the Classroom uses its plan representations in its interactions with the speaker. Each plan represents a common understanding of how a speaker and the Classroom should interact when the speaker pursues a particular course of action. The Classroom's side of the interaction usually involves controlling audio/visual components and filming what the speaker does. The speaker's actions prompt the Classroom to determine how it ought to cooperate.

Based on how the process monitoring and plan recognition work in the Classroom, there are two main issues that it must address to be effective in its cooperation with the speaker. First, the Classroom needs a way to choose a single plan to use in cooperation, when faced with potentially many candidate explanations. Second, the Classroom needs a way to ensure that its processes stay in synch with the speaker's.

4.3.1 Generalizing the set of candidate explanations

The first issue (of selecting a single plan from the potentially many) is a result of an important characteristic of the Classroom domain: the Classroom often needs to take action before it knows precisely what the speaker is doing. For instance, when the speaker heads over to the marker board, the Classroom does not know exactly what he is going to do when he gets there. He may write something, point at something, or erase something. Regardless of what he plans to do, the Classroom needs to continue filming the presentation. But, based on which he ends up doing, the camera techniques used may eventually differ.

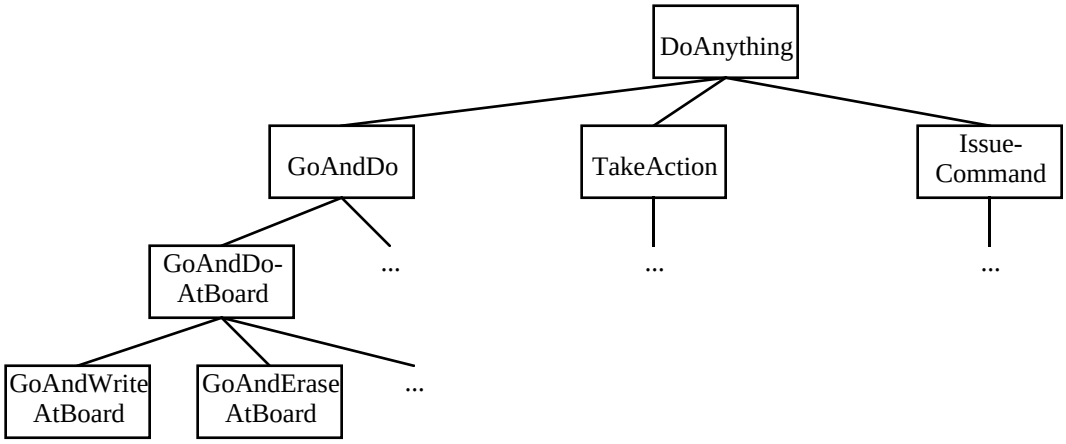


Figure 4.3: A portion of the Intelligent Classroom's speaker process hierarchy

The Classroom cannot simultaneously cooperate with every possible explanation for what the speaker is doing. Usually, trying to do so would require the Classroom to take contradictory actions (e.g. aiming the presentation camera in two different directions or both staying on the current slide and going on to the next). The Classroom's approach to addressing the problem is to make a more general explanation of what the speaker is doing, and then cooperate with that explanation. So, when the speaker approaches the board, the Classroom infers simply that he is going there to do something. The Classroom has an explanatory plan involving this vague action, and pursues it.

Figure 4.3 shows a portion of the Classroom's top-level process hierarchy. At the top of the hierarchy is the all-encompassing process of "doing anything". Below that are processes for the three main classifications of processes in the Classroom: (a) going somewhere and doing something when you get there, (b) doing something where you currently are, and (c) issuing some sort of a command (as either a spoken

command or a gesture). In plan recognition, the Classroom's initial explanation will often be one of these three processes. Below each of these three processes are both slightly more specific (but still general) processes and the actual particular processes that the speaker might be pursuing.

The Classroom uses this hierarchy to select a single process that is representative of the set of candidate explanations, using the following technique. First, it adopts the first element of the set as a tentative representative, to be refined using the remaining elements. Then, for each candidate explanation, it checks to see whether the current representative process is a generalization of the candidate. If so, this representative process adequately represents this additional explanation and is left unchanged. If it is not a generalization, then the representative process is further generalized until it is a generalization of the additional explanation. After this generalization has been performed for each candidate explanation, the Classroom adopts the plan associated with this representative process as its explanatory plan.

So, in the simple case where the Classroom has two candidate explanations (say, that the speaker is going to the board to write and that the speaker is going to the podium to lecture), it will adopt the representative process that the speaker is going somewhere to do something. Its initial representative process is that the speaker is going to the board to write, but in considering the second candidate explanation, the Classroom will first generalize the representative to the process for the speaker going to the board to do something, and then, since that is not a generalization of the podium process, further generalize it to the process for the speaker going somewhere to do something.

Once the Classroom has its set of candidate explanations and has selected a representative process for them, the Classroom will adopt the associated plan for the representative process, beginning execution of its processes in this plan. At the same time, the Classroom continues to monitor all of its candidate explanations and, as the explanations are rejected, it refines the representative process (and its associated plan). That is, as the Classroom gets a better idea of what the speaker is doing, it changes its actions appropriately. Whenever a candidate explanation is rejected, the Classroom finds a new representative process for the remaining explanations and it then adopts the associated explanatory plan.

Because the speaker and the Classroom are executing their processes in these plans while the plan recognition is going on, the Classroom cannot just restart the more specific plans and processes whenever the representative process is refined. Instead, the Classroom needs to start the plans and processes where the old ones left off. The Classroom accomplishes this through the structure of the plans (and their processes) in its plan library. For example, all speaker processes for "go somewhere and do something" plans have steps labeled `_go` and `_do`. The more specific processes may have additional steps, but all of the "go and do" processes share this bit of structure. Then, if the representative process is refined from "go somewhere and do something" to "go to the board and write" while the speaker is in the `_do` step of his process, the Classroom will know to start monitoring the new speaker process starting in the step labeled `_do`.

4.3.2 Synchronizing the processes in a plan

The synchronization clauses in a plan definition tell the Classroom how to fit its processes together with the speaker's in order to make the plan succeed. Each synchronization clause specifies a temporal relationship that must hold between two process steps (or in general, two processes). In the plans in the Classroom's plan library, one step is always in the speaker's process, while the other is in one of the Classroom's processes. So, looking back at the plan definition in Figure 4.1, the synchronization clause `(starts (_p1 _go) (_p2 _go))` specifies that the "go" steps of the two processes must start at the same time, and further that the speaker's process (i.e. the first one) must last at least as long as the Classroom's.

The Classroom uses the event mechanisms of the Process Manager to accomplish this synchronization. When the Classroom starts a plan, it not only starts the processes associated with the plan, but it also begins waiting for the events associated with its synchronization clauses. For instance, in the `starts` clause above, the Classroom will be waiting for the event of the speaker's process starting the step labeled `_go`. When this event occurs, the Classroom will then advance its process to its step labeled `_go`. The Classroom does this by attaching an event handler to its process that is waiting for the event corresponding to the speaker starting his "go" step and that has a result of advancing to the appropriate step. The Classroom adds one or two event handlers for each synchronization clause–a temporal constraint may constrain both the starting and stopping of a process step–and through the normal running of the Process Manager, the desired synchronization takes place.

Speaker:	**Classroom:**
1) "Play the video segment"	
	A) Cue the tape
	B) Set video source
	C) Start the VCR
2) "Pause"	
	D) Pause the tape
3) "Rewind a bit"	
	E) Slowly review
4) "Play"	
	F) Resume playing

Figure 4.4: A timeline of events occurring while the speaker presents a video segment

4.4 Fitting it all together

In this section, we look at how the plan representation discussed in this chapter is interpreted by the Intelligent Classroom. We step through the execution of two plans that are used in the Classroom, looking at how the plan and process pieces fit together. Each example starts at the moment that the Classroom realizes it needs a new explanation for what the speaker is doing and ends when the plan has reached its end.

4.4.1 Playing a video segment

Figure 4.4 shows a timeline representing the events that occurred during one instance of the speaker playing a video segment as a part of a presentation. The numbered events on the left refer to actions that the speaker took and the lettered events on the right refer to actions that the Classroom took in cooperation. Throughout this section, phrases such as “Event 1” and “Event A” refer to the corresponding events

```
(define-process (present-video-segment)
  (main-actor
   (person ?speaker))
  (set-up
   (require vcr-command-set))
  (steps
   (_1 (wait-for (speak-command i-start-vcr)
                 _play))
   (_play (wait-for (speak-command i-pause)
                    _pause))
          ...
          (wait-for (speak-command i-stop)
                    :done))
   ...))
```

Figure 4.5: A partial process definition for the speaker presenting a video segment

in this timeline.

For this example, the speaker already placed a videotape in the Classroom's VCR and furthermore, he used a presentation script (the special process that a speaker can use to outline the key events in his planned presentation) to specify what video segment is appropriate at this point in the presentation. (See Section 3.3 for more information about presentation scripts.)

Figure 4.5 shows the process making up the speaker's part in showing a video segment as a part of a presentation. The first step requires the speaker to command that the video be played and the other steps correspond to various states that the VCR can be in. The plan for showing a video segment includes this process and a process for the Classroom that directly controls the VCR. Each step in the speaker's process corresponds to a step in the Classroom's process that puts the VCR in the appropriate state. (There is also a Classroom process that is responsible for making the video feed of the presentation, but it will be ignored in this example.)

Referring back to the timeline in Figure 4.4, when the Classroom hears the speaker

say, "Play the video segment," (Event 1) it recognizes this as a command to go on to the next step in its presentation script and starts up the plan for showing a video segment (Event A). In starting its process in this plan, the Classroom makes sure the tape is cued properly (Event B), sets the display to accept the VCR output (Event C), and starts the videotape (Event D). From the speaker's process, the Classroom knows to start listening for the various voice commands involved in controlling a video presentation. (The fact that a video is playing provides a context in which to understand otherwise ambiguous commands like "stop" and "pause".)

Now both processes are in the steps corresponding to simply playing a videotape segment. The Classroom expects either that the speaker will say one of the VCR commands or that the segment will be allowed to play to completion. When the speaker says, "Pause," (Event 2) the speaker's process is advanced to the "pausing" step and, since the current plan requires that this step in the speaker's process be run at the same time as the corresponding step in the Classroom's process, the Classroom's process is also advanced to its "pausing" step, and the Classroom pauses the tape (Event E). In dealing with the remaining speaker events, the Classroom keeps its process synchronized with the speaker's, allowing the speaker to control the VCR.

4.4.2 Writing on the board

In this example, we focus on how the Classroom revises its understanding of what the speaker is doing over the course of several actions. We track how the Classroom builds a set of candidate explanations for a speaker's unexpected action and how the Classroom eventually discovers that the speaker is going over to the marker board to write something.

Figure 4.6 shows a timeline representing the events that occurred during one

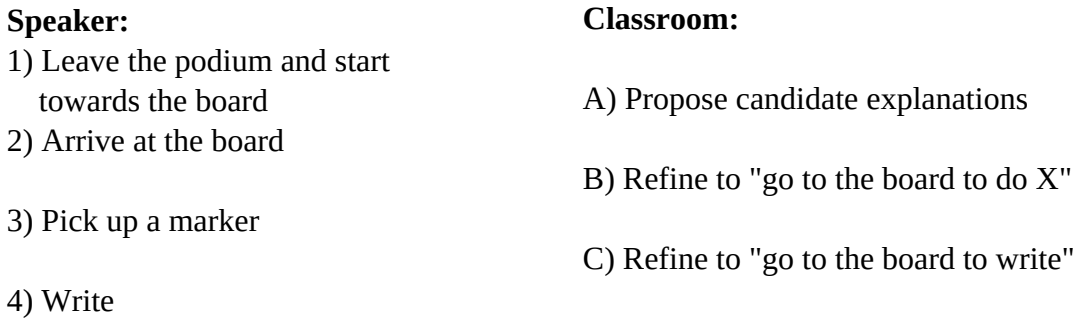


Figure 4.6: A timeline of events occurring while the speaker goes to the marker board and writes

instance of the speaker going over to the board and writing. In this example, the Classroom is not explicitly expecting the speaker to write on the board; there is no presentation script and the speaker is currently speaking at the podium. When the speaker leaves the podium (Event 1), the Classroom recognizes that this is an unexplained speaker action, requiring a new explanation for what the speaker is doing.

Unfortunately, in proposing candidate explanations (Event A), the Classroom finds that the speaker's single action does not restrict the speaker's possible plans very much. All the Classroom can infer is that the speaker is going somewhere to do something. Figure 4.7 shows a portion of the plan hierarchy with the candidate explanations highlighted. The links leading up to the generalized explanation are also highlighted. The Classroom adopts this overly general explanation (Event B) and starts up its process in this plan (which films the speaker as he walks about).

However, as the speaker moves towards the marker board, he is simultaneously moving away from the other potential destinations, causing plans that deal with the display screen or classroom props to be rejected. So, the Classroom infers that the speaker is going to do something at the board (Event B) and adjusts its presentation

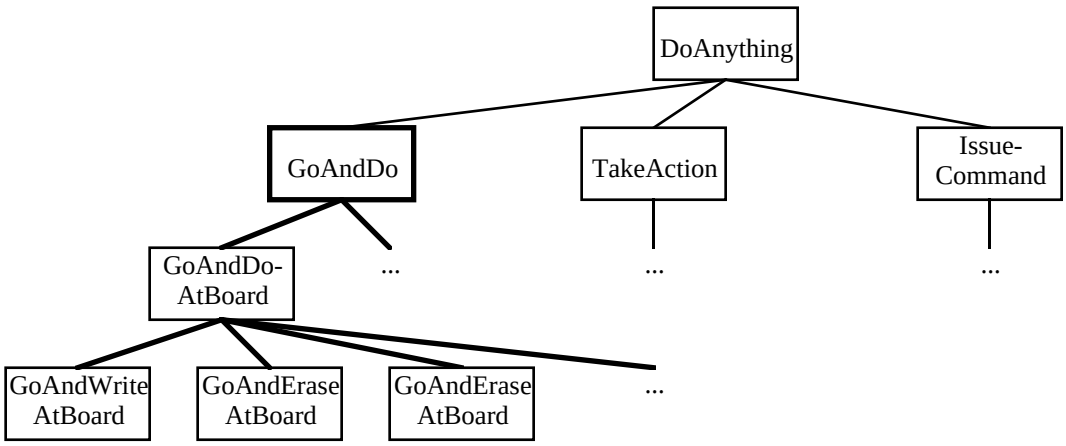


Figure 4.7: A portion of the Intelligent Classroom's speaker process hierarchy with links leading up to the generalization process chosen when the speaker starts moving

camera to focus on the board, even before the speaker arrives there (Event 2). Again, Figure 4.8 shows a portion of the plan hierarchy and highlights the set of candidate explanations at this moment.

Once the speaker is at the board, the Classroom is considering only three explanations for what he is doing: that he intends to write something, erase something, or point out something that he previously wrote. When he picks up a marker (Event 3), the Classroom finally recognizes precisely what he is doing (Event C) and begins to watch for writing motions. Figure 4.9 shows how the Classroom films the speaker as he writes. When the speaker starts writing, the Classroom keeps track of the region of the board that he writes on and focuses the camera there. Recall from 3.8 that the Classroom views the speaker's writing activity as a series of marker strokes. Each time the speaker writes, the Classroom maintains a bounding box that contains each of these strokes. These bounding boxes are stored, keeping track of all of the regions

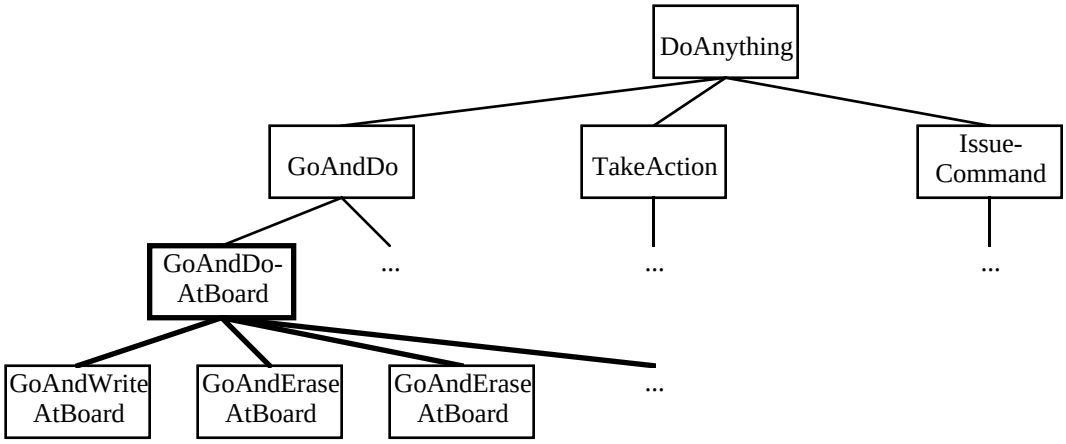


Figure 4.8: A portion of the Intelligent Classroom's speaker process hierarchy with links leading up to the generalization process chosen when the speaker approaches the board

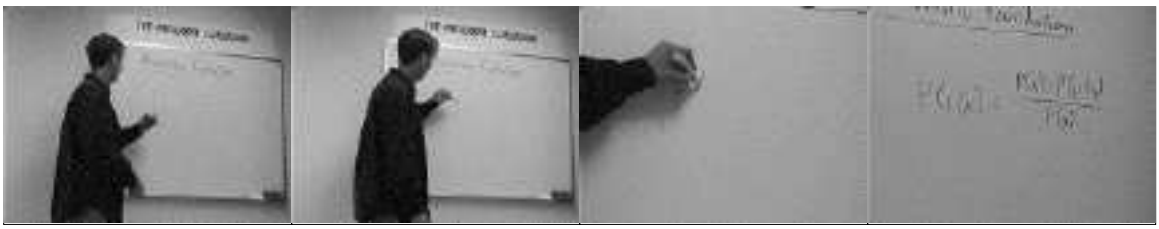


Figure 4.9: As the speaker writes, the Classroom zooms in on the writing

of the marker board that the speaker has written on. When the speaker starts writing, the Classroom can then tell if he is revising something he previously wrote or writing something new. Also, when the speaker points at something on the board, the Classroom can focus the presentation camera on the piece of writing he is actually referring to.

4.5 Open issues

The Intelligent Classroom uses the techniques described in this chapter to cooperate with the speaker in the sorts of tasks outlined in the examples. I believe that these techniques are sufficient to produce a reasonably competent A/V assistant for the Classroom. However, there are a few techniques that seem promising as a means for enhancing the Classroom's ability to cooperate. First, there are many possible improvements to the Classroom's plan generalization technique that would allow it to decide what the speaker is doing sooner. Second, there are some sorts of cooperative behavior that the Classroom could engage in that are not easily accomplished using the techniques in this chapter. Finally, I need to address potential complaints about the Classroom's view that a speaker is only doing one thing at a time.

4.5.1 Alternatives to the generalization method

When the Classroom is choosing a representative process for its set of candidate explanations, it makes the safest possible choice: if the speaker's actual activity is in the explanation set, the Classroom's representative process is guaranteed to be a generalization of it. But, while using this technique helps produce reasonable behavior in almost all circumstances, the Classroom ignores any notion of likelihood or of the cost of being wrong or acting late. These factors, if employed well, could enable the Classroom to be more responsive to the speaker's plans as it chooses more specific representative processes.

The notion of likelihood would come into play in encouraging the Classroom to gravitate towards the more probable explanations and to avoid the less likely. If the Classroom keeps track of how often the speaker pursues each plan, it can, for any

set of candidate explanations, compute how likely it is that the speaker is actually pursuing any given explanation. Then the Classroom could ignore the less probable explanations when choosing its representative process. For example, if the Classroom was willing to over-specialize five percent of the time, it could sort the candidate explanations by probability and then ignore the least likely explanations whose probabilities sum to five percent. Better still, the Classroom could filter the explanations while choosing its representative process, recognizing that only a few of the explanations may have an effect on the generalization process and therefore choosing to ignore the least probable of these.

Another important consideration in choosing a representative process is how bad it is to make a poor choice. For example, the Classroom's camera-control processes in the more general explanatory plans tend to utilize wider camera angles than the more specific ones. As a result, choosing an overly general representative process usually is fine, as far as controlling the camera. However, when close camera shots are needed (e.g. when the speaker writes on the board or points at something), an overly general representative process can reduce the quality of the video-feed. And, potentially worse, if the Classroom includes likelihood in choosing a representative process, there is the possibility that the Classroom will choose a specific but incorrect explanation.

4.5.2 Broader notions of cooperation

Currently, the Classroom's model for cooperation is something like this: (a) recognize what the speaker is doing, (b) identify the particular plan appropriate to the speaker's actions, and (c) cooperate by executing the appropriate processes in the plan and staying in synch with the speaker. This model fits a wide range of the sorts of

cooperative behavior that are appropriate in the Intelligent Classroom domain, but there are a few classes of cooperation that it does not fit.

There may be situations where it would be appropriate for the Classroom to solicit help from the speaker. For example, if one of the Classrooms components (e.g. a VCR) has been inadvertently turned off, the Classroom will not be able to turn it on itself and so, if the component is needed in the presentation, it would be appropriate for the Classroom to ask the speaker to turn it on. The Classroom can recognize many such problems and, in some situations, it may be reasonable for the Classroom to talk the speaker through some simple troubleshooting steps. For these sorts of Classroom-initiated cooperation, the Classroom's basic plan and process algorithms and representations would support the actual cooperation–the Classroom would just need some principled way of deciding when to ask for help.

Also, there may be situations where the Classroom can assist the speaker by recognizing when something is wrong, before the speaker even realizes it. For example, if the speaker goes to the board and selects a pen that is out of ink or if the speaker asks the Classroom to play a video segment when there is no tape in the VCR, the Classroom knows immediately that something is wrong: some state of the world that is required for the successful completion of the speaker's plan does not hold. In some circumstances it may be appropriate for the Classroom to alert the speaker. The Classroom could identify many of these sorts of problems, as it starts up all the processes involved in the speaker's plan; the Classroom would fail to bind some of the process roles or other variables. However, this is potentially a very annoying sort of "cooperation" and the Classroom would need to be cautious not to needlessly interrupt the speaker due to inconsequential problems.

4.5.3 Is the "single speaker plan" hypothesis okay?

In discussing the Classroom's plan recognition techniques with other researchers, I have heard many people doubt that the Classroom's insistence upon having a single goal is adequate to capture how speakers typically present a lecture. Instead, they point out the speakers tend to have multiple goals that they are pursuing and that they often interleave the actions of their plans in arbitrary ways. I am led to believe them–in my own lectures, I often find myself doing this very thing. Fortunately, in the Classroom, this potential problem disappears because, despite what is going on inside the speaker's head, the Classroom can still interpret the resulting presentations as if they were actually linear. For example, if the speaker is interleaving a slide presentation with a series of updates to a figure at the marker board, the speaker probably views his action as two processes running in parallel, but the Classroom can just as easily see the action as a series of trips between the podium and the board.

Having said that, the "single speaker plan" hypothesis would probably limit the cooperative ability of agents in many other domains. The difficulty, of course, is that by introducing the possibility of the speaker performing more than one plan at time, we also make the plan recognition much more difficult. In trying to explain three speaker actions, the Classroom must consider that the actions were from one plan, two plans, or even three plans; there are five possible ways of grouping the three actions into plans that explain them. In [51], these difficulties are discussed in great detail and dealt with to a degree by making simplifying assumptions like always preferring an explanation with the fewest number of constituent plans.

The Classroom, through the use of presentation scripts, does allow the speaker to do a limited amount of multi-tasking. When the speaker deviates from his script, the

Classroom suspends it, waiting for the speaker to resume his planned presentation. But, while this does do a better job of explaining some speaker behavior, it still falls short of actually understanding what the speaker is doing when he is running back and forth between the podium and the board.

Chapter 5

Jabberwocky

In this chapter, we look at a project that grew out of the Intelligent Classroom: the Jabberwocky slide-show controller. This project is of particular interest because it uses the representations and algorithms of the Classroom as a starting point and then extends them to deal with the slightly different demands of running a slide presentation. The probabilistic techniques used in Jabberwocky to deal with the unreliable speech recognition results may be adapted to improve the Classroom's ability to deal with its own sensor noise. Also, the simple learning techniques used in Jabberwocky capitalize on the speech recognizer's tendency to make mistakes in the same way (e.g. if the speech recognizer mishears a word in a particular way once, it is likely to mishear it that way many other times). The Classroom's computer vision system also often errs in the same way (e.g. the speaker's head will often momentarily "disappear" whenever the speaker walks across a shadow on the wall). Hopefully the learning techniques developed for Jabberwocky will also be applicable to the Classroom to deal with its own sensor noise.

5.1 Foundations

Jabberwocky was developed in response to the following question. What should the Classroom do in incorporating slide-based presentations? Naturally, the Classroom must allow the speaker to use commands and gestures to control the presentation. But also, it would be very useful to have the Classroom look at the content of the slides, matching the words and phrases the speaker says to words and phrases in the slides, and allowing the speaker to control a slide presentation by simply lecturing. Essentially, the Classroom follows along with the speaker as he discusses the current slide and then the Classroom switches slides when it determines that he has moved on to another slide. By combining the command-based and content-based approaches, Jabberwocky is able to support a wide range of presentation styles, allowing speakers to present their slides in whichever manner they feel will be the most effective. For example, the content-based approach lets the speaker skip around in his presentation by describing the slide that he wishes to skip to–this would prove invaluable for answering questions from the audience. In the extreme, this could even support a speaker who wished to do an unstructured lecture from a large body of slides.

5.1.1 How do people want to lecture?

Our primary goal as we designed and implemented Jabberwocky was to construct something that we would actually use. To address this, we needed to envision how such a system would be used and then make sure that the system's requirements were reasonable to attain. We did not wish to invent new ways of lecturing but instead to enable people to utilize slides in their preferred lecture styles.

One curse of overhead transparencies, slides and PowerPoint is that the speaker

is essentially forced to give his presentation in precisely the same way every time. To deviate from the prescribed order, he must rifle through his set of overhead transparencies, skip over all the slides (one at a time) between the current slide and the desired one, or, in PowerPoint, select the slide in a menu. As a result of this inconvenience, speakers tend to let the slides dictate the presentation instead of supporting it. Of course, there are times that this rigid structure helps, rather than hinders a speaker. For example, a nervous or scatterbrained speaker may appreciate having the set of slides provide a structure to his talk. But, in many cases, a speaker would prefer to have more flexibility in giving his presentation.

There are, of course, speakers who refuse to be caged. A speaker from our laboratory once gave a twenty-minute conference presentation using a box of more than one hundred overhead transparencies. He had painstakingly organized them into at least twenty categories and did numerous practice talks to become proficient at extracting slides to respond to particular questions. During his talk, an audience member asked one of the questions that he had prepared a slide for, and he quickly located the appropriate slide and displayed it. As the audience murmured its admiration, he said, "I'm glad you asked that!" However, the vast majority of speakers are not willing to go to such great lengths to prepare for a presentation–nor should they have to be.

With PowerPoint (and most similar presentation software), speakers may use hypertext links within their slides to create flexible presentations. However, this again requires a great deal of preparatory effort and also forces the speaker to use the computer's mouse for much of his navigation through his lecture. The point here is that although it is possible to give flexible presentations using slide-based media, almost no one even tries to, being dissuaded by the effort required to do so.

Perhaps it is not surprising that many of the most dynamic and spontaneous teachers avoid using slides in their presentations. They wish to address their students' questions or reorganize their lectures on the fly to meet their particular students' needs. They wish to explore interesting tangents when the lecture provides an opportunity. It is very difficult to do these sorts of things in the confines of a traditional slide-based presentation. With Jabberwocky, we hope to support these kinds of spontaneity in a slide-based presentation.

5.1.2 The basic operation of Jabberwocky

Basically, Jabberwocky (the Classroom and the stand-alone incarnation) uses the speaker's slides as a rough outline. As the speaker goes about his presentation, Jabberwocky listens for clues as to where the speaker is in this outline. Usually, the clues confirm that he is on the slide that Jabberwocky is displaying, but occasionally, the clues indicate that Jabberwocky is not on the correct slide. These clues take one of two forms: (1) words and phrases that are indicative of particular slides and slide points and (2) commands that indicate how the speaker wishes to move through the presentation.

Before the speaker begins his presentation, Jabberwocky analyzes the slides and constructs a hierarchical set of processes for following along with the presentation. The steps in the top-level process correspond to individual slides, and each slide's process has steps corresponding to its key points. The steps in these slide processes are essentially interpreted as saying things like, "if you hear the speaker say 'probability distribution' then he is probably discussing the third point in this slide." The idea is that, if Jabberwocky is able to choose these clues well, it will have enough information to keep track of the speaker's place during the presentation.

Once the presentation has begun, Jabberwocky listens for the set of clues appropriate to its current place in the presentation. It will listen for the key words and phrases associated with the current slide (to keep track of which point the speaker is on) and the possible next slides (to see when the speaker has gone on to another slide), and it will also listen for the various commands that are appropriate to the slide presentation. As the speaker moves through the slide presentation, Jabberwocky changes the words and phrases that it is listening for.

Commands allow speakers direct control of their presentations. This level of control may be needed to remedy mistakes that Jabberwocky makes (e.g. switching slides too early or too late). Also, this level of control may be useful when indirect (i.e. content-based) control is awkward or infeasible. For example, while a well-rehearsed speaker may have no difficulty leading Jabberwocky, a less confident speaker may wish to allow Jabberwocky to lead him. Such a speaker often will not be certain of what is on the desired next slide, and needs to command Jabberwocky to go to the next slide in order to find out. In addition, almost any speaker would prefer saying, "skip ahead to the conclusion slide" to trying to describe its contents. The various commands used in Jabberwocky will be discussed as appropriate through this chapter.

5.1.3 Extracting key phrases from slides

To be successful in its content-based slide advancing technique, Jabberwocky must do a good job of extracting key phrases from the slides–these phrases serve as Jabberwocky's clues as to where the speaker is in his presentation. This shallow understanding is in lieu of a rigorous natural language understanding of the presentation, which is simply not feasible for Jabberwocky. Fortunately, this deeper understanding is not necessary for Jabberwocky; a human audio/visual assistant can match what the

‘Twas brillig, and the slithy toves
  Did gyre and gimble in the wabe:
All mimsy were the borogoves,
  And the mome raths outgrabe.

"Beware the Jabberwock, my son!
  The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
  The frumious Bandersnatch!"

He took his vorpal sword in hand:
  Long time the manxome foe he sought --
So rested he by the Tumtum tree,
  And stood awhile in thought.

Figure 5.1: The initial stanzas of Jabberwocky with a drawing of the battle with the terrible beast

speaker is saying to the contents of the slides without any technical knowledge of the presentation’s subject knowledge. It appears that a purely syntactic understanding is sufficient for knowing what to do. It is like Alice’s situation after reading the poem Jabberwocky (shown in Figure 5.1) in Lewis Carroll’s “Through the Looking-Glass” [16]. The poem was nonsense to her, but she was still able to glean a little meaning from it: “Somehow it seems to fill my head with ideas–only I don’t exactly know what they are! However, somebody killed something: that’s clear at any rate.”

The primary way that Jabberwocky locates words and phrases that are representative of slides and slide points is through syntactic analysis. The idea behind the approach is that much of the meaning of a sentence can be grasped just by looking at the various actors and actions in the sentence. In discussing a slide point, the speaker can be expected to at least mention its actors and actions. They may be discussed

Phrase Rules for Noun Phrases:
(:adj* :noun+)

Phrase Rules for Verb Phrases:
(:adv* :vhelp* :adv* :v :adv*)

Key to Abbreviations:
:adj = adjective
:adv = adverb
:noun = noun
:v = verb (any form)
:vhelp = helping verb
* = zero or more
+ = one or more

Figure 5.2: The phrase rules Jabberwocky uses to locate noun and verb phrases in the slide text

in a different order than they appear in the slide (e.g. changing a sentence from the passive to the active voice). The speaker may even dramatically change how they are described (e.g. converting adjectives into prepositional phrases). These potential problems are dealt with by choosing phrases to independently represent the actors and actions and by creating numerous synonymous phrases for actors and actions that can be expressed in many ways.

Jabberwocky locates phrases for the important actors and actions in a sentence by first finding all the noun and verb phrases, and then removing "uninteresting" words from them. (Figure 5.2 shows the rules used to locate the phrases.) Articles, pronouns, propositions and words that essentially serve as syntactic glue are among the words that Jabberwocky filters out. Jabberwocky considers the remaining short sequences of words (and the paraphrases developed using them) as representative phrases for the sentence. These techniques for selecting important words and phrases have also been employed on Rosetta [10], a system that indexes research papers based on how they have been cited, and on DRAMA [78], which uses free-form text in its indices for case-based retrieval.

Transformation Rules for Noun Phrases:

(:noun1)	(:noun1)
(:noun1 :noun2)	(:noun1) (:noun2) (:noun1 :noun2)
(:adj1 :noun1)	(:noun1) (:adj1 :noun1)
(:adj1 :noun1 :noun2)	(:noun1) (:noun2) (:adj1 :noun1)
	(:adj1 :noun2) (:adj1 :noun1 :noun2)

...

Transformation Rules for Verb Phrases:

(:adv1 :v1)	(:v1) (:adv1 :v1) (:v1 :adv1)
(:vhelp1 :v1 :adv1)	(:v1) (:adv1 :v1) (:v1 :adv1)
	(:vhelp :v1) (:adv1 :vhelp1 :v1)
	(:vhelp1 :adv1 :v1) (:vhelp1 :v1 :adv1)

...

Figure 5.3: The syntactic transformation rules Jabberwocky uses to construct synonymous noun and verb phrases

Jabberwocky also looks at non-text objects in the slide when creating these key phrases. In slides containing graphs, tables or pictures, Jabberwocky can anticipate that a speaker will mention these slide elements while discussing the slide. So, for a slide containing a table, Jabberwocky will listen for phrases like "in this table" and "in the table on the right" in addition to the phrases derived from the slide's text content. For each of the different types of object we expect to find embedded in a slide, we have enumerated a number of key phrases that could be used refer to it.

When designing slides, speakers tend to be as concise as possible–putting as much content as possible into just a few words. As a result, slides are often full of convoluted language; overly complicated noun and verb phrases abound. But, when speaking, people tend to use more natural language, avoiding the stilted prose of their slides. Therefore, Jabberwocky cannot assume that speakers will describe actors and actions

in the slide contents using exactly the same words as the slides do. So, for each phrase, Jabberwocky constructs a number of synonymous phrases, using its set of syntactic transformation rules. (See Figure 5.3 for a partial list of these transformations.)

With verb phrases, Jabberwocky constructs a phrase for each way the adverbs and helping verbs can be moved or eliminated. From the verb phrase "can immediately advance", it also gets phrases like "can advance immediately", "immediately advance" and, simply, "advance". Also, in applying these transformations, Jabberwocky converts all the nouns and verbs into their root forms. Therefore, the derived phrases also account for verb conjugation changes due to tense, plurality or use of the passive voice.

With noun phrases, Jabberwocky breaks the phrases into smaller units: the adjectives, the modifier nouns and the subject noun. From the noun phrase "current information management technology" Jabberwocky gets "current", "information management" and "technology" for the three units. Given these units, Jabberwocky constructs key phrases using the following rules:

- The modifier nouns can serve as a key phrase (as in "information management").
- The modifier nouns can be separated from the adjective and subject noun (as in "current technology for information management").
- Different subsets of the adjectives and modifier nouns can be used (as in "information technology").

Finally Jabberwocky takes one, two and three word subsequences of the resulting phrases–one-word phrases are more tolerant of words being missed by the speech recognizer while three-word phrases are much more predictive of what the speaker

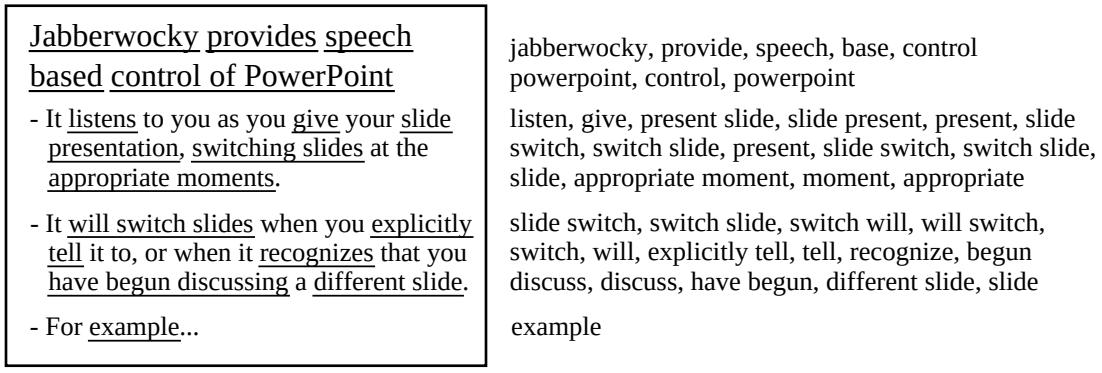


Figure 5.4: Phrase extraction in Jabberwocky. On the left, a slide, with identified phrases underlined. On the right, the words and phrases that Jabberwocky will be listening for.

is actually talking about. Using these rules for converting noun and verb phrases into a large number of words and phrases, Jabberwocky is able to anticipate how the speaker will paraphrase his slides, without diluting the slide's meaning so much that it makes wishful false matches.

In Figure 5.4, we see the verb and noun phrases that Jabberwocky identified in an example slide (on the left) and the words and phrases that Jabberwocky derived from them (on the right). It is important to note that the phrases on the right are made up of words in their root forms and also that they omit certain connecting words like prepositions and possessives. When Jabberwocky is listening for these phrases, it will also strip out these same words when matching them to the speaker's words. So the derived phrase "present slide" would be found in the spoken phrases "when you are presenting slides" and "in presenting your slides" among others.

5.2 As a part of the Intelligent Classroom

The original incarnation of Jabberwocky was a slide-switching component in the Intelligent Classroom. While this version was quickly abandoned in favor of the standalone version described in the next section, the Classroom incarnation is important in considering how the basic Classroom representations and algorithms can be extended. In the case of the evolution of Jabberwocky, we kept the basic ideas of treating a slide show as a presentation script and treating speech commands as speaker actions that must be acted upon. But, in the final version of Jabberwocky, we ended up utilizing a probabilistic approach to deal with the individual words and phrases that Jabberwocky hears the speaker say. This change proved necessary because the speech recognition results are so unreliable in realistic presentation environments.

In this section, we discuss the implementation of the original incarnation of Jabberwocky. We start with the sensors and actuators used in listening to and watching the speaker and in displaying the different slides and then look at how Jabberwocky transformed a slide show into a set of processes for running the actual presentation. In the current Intelligent Classroom implementation, the probabilistic approach has been applied to the techniques described here.

5.2.1 Sensors and actuators

Recall that the Intelligent Classroom's skill system connects the Classroom's Process Manager to its physical sensors and actuators. In the case of Jabberwocky, the important sensors are its speech recognizer (listening for words, phrases and commands) and its vision system (watching for gestures to be treated as slide-show commands). The sole actuator is its slide-show controller, which displays particular slides on the

Classroom's main display.

For speech recognition, we use IBM's ViaVoice software [46], a commercial speech recognition product that featured a powerful, yet easy to use, programmers API. ViaVoice supports two modes of voice recognition: continuous dictation and command-based. In the continuous dictation mode, ViaVoice attempts to recognize every word that is said, using the surrounding words as clues to help with ambiguous words. The software promises 95% accuracy, which can be achieved when the system is trained and the speaker is consciously over-enunciating. When a speaker talks conversationally, the accuracy can easily fall below 50%. In the command-based mode, ViaVoice listens for specific commands. The command language is specified using a context-free grammar. The software expects commands to be preceded and followed by brief pauses–this is how commands are distinguished from dictation. The accuracy of the command-based mode seems very good.

In the Classroom, we perform speech recognition on a dedicated machine to improve responsiveness; when the speech recognizer gets bogged down on a busy computer, it cues its audio input and its results can lag fifteen or more seconds behind the actual utterances. This computer runs a simple speech recognition server, which sends all of the recognized words and commands to the Classroom's main computer via a TCP stream connection. This stream is read by a Classroom skill that reports all of the utterances to the Process Manager as events. The commands map quite naturally into speaker events; all of the utterances that are treated as "next slide" commands cause the skill to report the same "the speaker wants to go on to the next slide" event.

Reporting events corresponding to the words and phrases that the speaker says is

Figure 5.5: The three slide-control gestures: touch the next slide button, point right and point left.

somewhat more difficult. The skill must extract phrases from the (continuous) stream of recognized words in such a way that Jabberwocky will get the ones it is waiting for. It first filters the same non-content words that Jabberwocky did (in deriving its key phrases) and then converts the remaining words into their root forms. Then, rather than reporting all one, two and three word subsequences of the derived word stream, the skill only reports phrases that processes in the Process Manager are waiting for. The skill registers with the Memory System ("snoop" functions are discussed in Section 3.4.2), indicating that it is interested in events that the processes are waiting for. Then, the events dealing with phrases update a data structure that the skill uses to efficiently identify when the speaker has uttered a phrase that Jabberwocky is waiting for.

For gesture recognition, the Classroom starts up skills that identify the three gestures that a speaker can use to control Jabberwocky: pointing to his right to indicate he wishes to show the next slide, pointing to his left to indicate he wishes to review the previous slide, and touching the next slide "button" on the display screen to advance the slide. The speaker can perform the first two gestures anywhere in the

Classroom; the Classroom waits for him to be stationary and hold one of his hands out to his side for a moment. For the third gesture, the speaker must be standing next to the display screen and then hold his hand over the button displayed as a part of the slide. Figure 5.5 shows a speaker performing each of these three gestures.

For Jabberwocky's sole actuator, the slide displayer, we made our own slide-displaying program, enabling us to have a great deal of control over how the slides appeared. In particular, we could easily highlight slide points as they were mentioned. (We would have preferred to simply use Microsoft PowerPoint, but, at that time, the programmer's API for Macintosh did not allow us the control we needed.) The slide displayer allowed Jabberwocky to select slides in any order and highlight points on the active slide.

5.2.2 Slide shows as presentation scripts

In Jabberwocky's original incarnation, it controlled the slide presentation by executing a presentation script it derived from the set of slides. Jabberwocky would transform them into a presentation script and a set of associated processes. The top-level presentation script controlled the flow of the presentation from one slide to the next and the other processes followed along with the speaker's progress through individual slides.

In building all the processes for a slide show, Jabberwocky would import the essential contents of the slide show, extracting the text of each slide's title and points and also noting whether each slide contained embedded pictures, graphs or tables. For each slide, it would locate the key phrases for the title, points and embedded objects and then define a process for the slide that contained steps corresponding to the title and each individual point. Each step would be waiting for the speaker to say

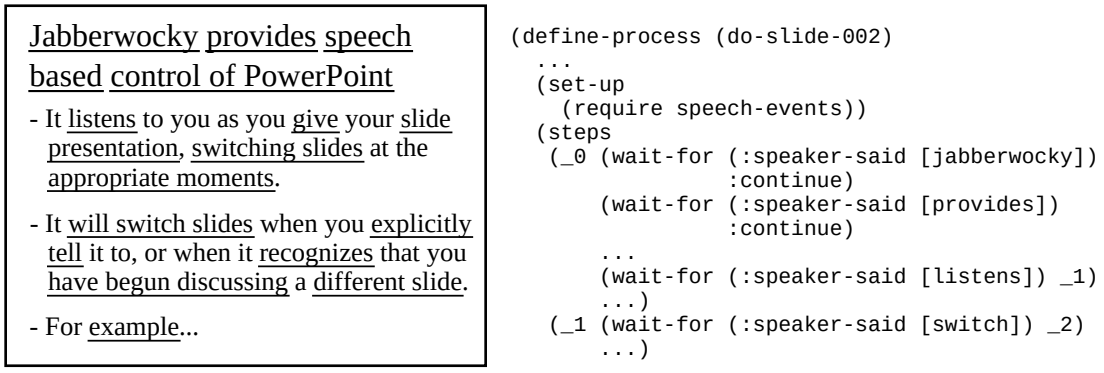
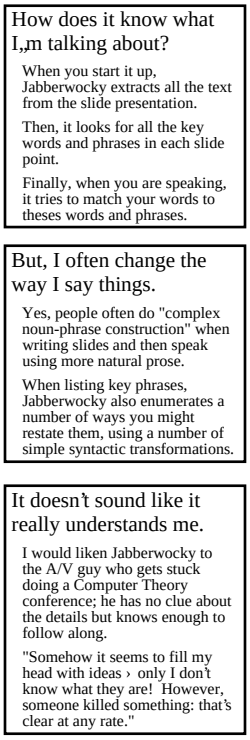


Figure 5.6: A slide and a portion of the process representation used to follow along with it. The initial wait-for clauses in step _0 are used to identify the slide when there is a branch point in the slide presentation.

one of the phrases from the next point, causing the process to advance to the next step. When the process advanced, it would also cause the next point in the slide to be highlighted. Figure 5.6 shows a slide and the process that Jabberwocky generated to follow along with it.

Jabberwocky would then build a presentation script that used these slide processes. Each step in this presentation script corresponded to a slide and would execute the process for the associated slide and wait for events that indicated that the speaker had moved on to a different slide. The speaker could indicate slide changes through gestures and speech commands. Each step would wait for each possible command and move to the appropriate next step in response. Each step would also wait for the speaker to say words and phrases that indicated that he had moved on to the next slide. So, the step would include event handlers for each phrase in the next slide where that phrase was not also a part of the current slide. Figure 5.7 shows a set of slides and a portion of the presentation script for following along with them.




```
(define-process (slide-show-001)
  ...
  (steps
   ...
   (_3 (assert (slide-shown slide-003))
       (wait-for (:speaker-said [often change]) _4)
       (wait-for (:speaker-said [change]) _4)
       (wait-for (:speaker-said [things]) _4)
       ...
       (wait-for (:speaker-commanded i-next-slide) _4)
       (wait-for (:speaker-commanded i-go-back) _2)
       ...)
   (_4 (assert (slide-shown slide-004))
       (wait-for (:speaker-said [sound]) _5)
       (wait-for (:speaker-said [really understands]) _5)
       (wait-for (:speaker-said [understands]) _5)
       ...
       (wait-for (:speaker-commanded i-next-slide) _4)
       (wait-for (:speaker-commanded i-go-back) _2)
       ...)
   (_4 (assert (slide-shown slide-005))
       ...)
   ...))
```

Figure 5.7: A portion of the slides in a slide presentation and the part of the slide show presentation script used to follow along with them

To make presentations more flexible, Jabberwocky also allowed the speaker to provide a flow chart, specifying which slides might follow any given slide. This allowed a speaker to have decision points in his presentation where he would decide (during the presentation) whether to cover a particular section, or to skip over it. Then, for the steps in the presentation script that corresponded to these decision points, it would have to wait for phrases from every possible next slide.

5.3 As a stand-alone application

Before we had even completed the implementation of the Intelligent Classroom version of Jabberwocky, we began to see many problems that a working system would need to address. Many of these problems stemmed from the fact that the speech recognition results were substantially worse than we had anticipated; when a speaker was talking rapidly, the resulting speech accuracy often fell well below fifty percent. With such poor results, Jabberwocky would rarely hear two or three word phrases correctly, but the single word phrases that it did hear did not have strong predictive power since the words often appeared in numerous slides. In addition, the speech recognizer would occasionally falsely "hear" the speaker say a phrase that indicated they were moving on to another slide, prompting Jabberwocky to advance the slide mistakenly. Also, it was difficult to concisely encode such simple speaker tendencies as the notion that a speaker will usually address each slide point before going on to a different slide.

These problems led us to utilize probabilistic methods to make Jabberwocky more tolerant of speech recognition errors and better able to consider the ways that speakers tend to lecture from slides. In making these design changes, we also implemented Jabberwocky as a stand-alone application, running on the same machine as Microsoft PowerPoint and controlling the presentation directly.

5.3.1 Probabilistic Foundation

During the course of a typical presentation there are often situations where the speaker needs to deviate from the planned presentation: an audience member's question addresses a previous slide; or time constraints require the speaker to condense his presentation. Sometimes the speaker may not even have a planned presentation; he may

Bayes„ Law: $P(s|p) = P(s) * P(p|s) / P(p)$

"The likelihood that the speaker is lecturing from slide *s*, given that he just said phrase *p*, is the probability that he already wanted slide *s* times the probability that he would say phrase *p* (if he were lecturing from slide *s*), divided by the probability he would say phrase *p* (at any moment in his lecture)."

Figure 5.8: Bayes Law with an English (barely!) translation of how it is understood in controlling a slide presentation

have a body of slides that he wishes to use during an improvised discussion. In this section, we look at how the word and phrase "clues" (from Section 5.1.3), coupled with a probabilistic approach, can support these sorts of dynamic presentations.

Our probabilistic approach utilizes Bayes' Law to update a probability distribution across the set of slides. When Jabberwocky decides that the speaker may want to skip to another slide, it will first assign an initial probability to each slide (based on how likely it is that the speaker will skip to it). Then, after each word or phrase that it hears, it will update these probabilities in response (using Bayes' Law). Finally, when one slide clearly dominates, Jabberwocky switches to that slide. Figure 5.8 shows Bayes' Law and provides a description of how Jabberwocky applies the equation to switching slides. The s's refer to the indices of slides and the p's refer to the key words and phrases that have been extracted from the slide content. In particular:

- $P(s|p)$ is the probability that the speaker wants slide s given that he just said phrase p. We compute this using the other values.

- *P(s)* is the probability (immediately prior to hearing phrase p) that the speaker wants slide s. *P(s)* takes into account both how likely the speaker is to want

to change slides at this moment (based upon how much of the slide he has discussed) and which slides are likely to follow this one (based on probabilities dependent on the style of presentation). If the speaker has just started discussing a new slide, *P(s)* will be very high for that slide, and very low for the others. Then, as he discusses it, *P(s)* for that slide will gradually decrease, as *P(s)* for the possible next slides will increase.

- $P(p|s)$ is the probability that the speaker would say phrase p, if he wants slide s. If we wished to assume the speaker were to simply read verbatim from the slides, $P(p|s)$ would simply be the number of times phrase p appears in slide s divided by the number of phrases we have extracted from the slide. Since most speakers do not just read the slides, we also consider it possible (though less likely) that the speaker will use phrases that appear elsewhere in the presentation. (For most presentations there will be a number of important phrases that are spoken repeatedly throughout the presentation, but that will only appear in a few slides.) So, we compute $P(p|s)$ as a weighted sum of the probabilities based on phrases in the slide and phrases in the presentation.

- $P(p)$ is the probability that the speaker would say phrase p in the current situation. This is simply the sum of all the $P(p|s)$, weighted by $P(s)$.

In a situation where the speaker wants to skip to a particular slide, he will tell Jabberwocky (through a command) that he wishes to skip and then will begin discussing the desired slide. For example, the speaker might say "Please skip back $¡$pause$¿$ to the slide that talks about how Jabberwocky extracts phrases from the slide contents." By the time Jabberwocky has updated the probabilities for the phrases "extracts",

"phrases" and "slide contents" the probability for the desired slide will be sufficiently high that Jabberwocky will switch to it.

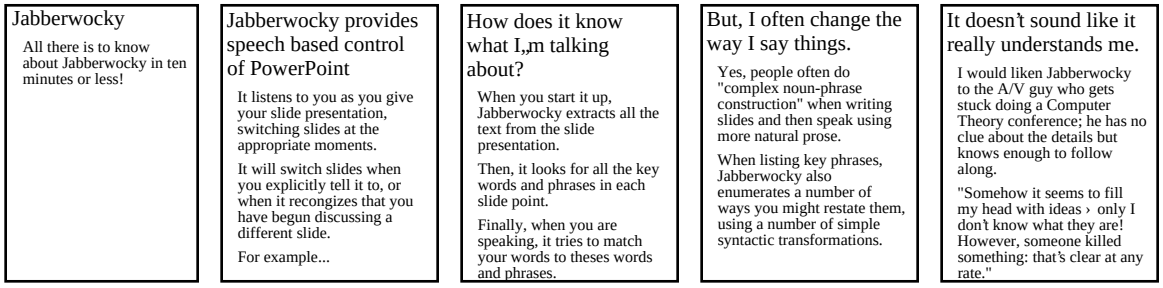
When the speaker makes a slide-skipping command, Jabberwocky must first determine which slides the speaker might lecture from next. Based on the particular command, Jabberwocky may be able to reduce the set of slides to consider. For example, if he says, "skip back..." then we need only consider slides that have been seen already. Jabberwocky distributes the probability evenly among the slides that it is considering.

Then, as the speaker talks, Jabberwocky listens for all the phrases it has extracted from the set of slides under consideration. When it hears one of these phrases, it computes $P(s|p)$ for each slide to update the probability distribution. Then if the probability for one slide is sufficiently high (90%), it will skip to that slide and continue the presentation from there. Otherwise, Jabberwocky will continue listening for phrases and updating the probabilities.

In a truly freeform lecture (where the speaker is treating his slides as a set–rather than a sequence), Jabberwocky will remain in the probabilistic mode indefinitely. In this case, the probability distribution is set up to favor slides that have not yet been viewed. But also, when updated, the probabilities are adjusted such that no slide becomes too improbable. We will need to do more practical experimentation to determine how to best constrain the probabilities and facilitate this mode of lecturing.

5.3.2 Following along with a free-form slide presentation

In this section we take a look at Jabberwocky in action. In this example, we show what happens as the speaker lectures from a set of five slides, looking at how the words and phrases extracted from the slides allow Jabberwocky to identify which slide the



<pause> today it all to you ever thing to one know about jabberwocky 10 minutes to less <pause> so for stop what does into <pause> the essentially provides a **speech** based interface to microsoft's power plant presentation software <pause> it listens is to presenters lads switching sides of the appropriate moments <pause> it switches the slide when it's possibly tell until when recognizes the jurors discussed in different <pause> for example <pause> new wonder how is the norm talking about <pause> while winning **started** up jabberwocky to extract the text premise slides in the presentation <pause> then defines key words and phrases and in each from <pause> finely when you speak it matches your words to these words and phrases <pause> blame myself and change the way say **things** <pause> and the people often construct complex now praises when making their slides and then in discussing them use more natural process <pause> when this thing key phrases jabberwocky also uses the number of samples and tactic transformations <pause> to enumerate the number of ways to my mistake them <pause> so you replied that doesn't sound **like** it really understands be all <pause> troops <pause> i would like in jabberwocky to the tv guide the computer theory conference <pause> his clueless regarding what's been talked about <pause> the still able to follow along <pause> hornet's like alice from lewis carroll said the looking glass after reading the jabberwocky <pause> she said somehow it seems to fill my head with ideas <pause> only at don't know what they are <pause> however someone killed some thing that's clear to anyone

Figure 5.9: Results from the first example. Across the top, the five slides that the speaker lectured from. On the left, what Jabberwocky heard the speaker say in his narration. On the right, the probability distribution during the course of the talk.

speaker is lecturing from. Figure 5.9 shows five slides (across the top) that informally describe the operation of Jabberwocky. Since this is a free-form lecture, Jabberwocky assumes nothing about the order of the slides, but uses the probabilistic method to determine what slide to switch to and when. In order to make the figure easier to understand, the slides happen to be shown in the order that they were lectured from.

On the left of the figure is a transcript of what Jabberwocky heard the speaker say. The words in bold indicate the places where Jabberwocky decided to switch slides. The underlined words mark phrases that Jabberwocky used as evidence for switching to another slide. On the right is a chart displaying the probability distribution as the talked progressed. Time progresses downward in the chart to approximately follow along with the transcript. The widths of the five vertical stripes at a particular height correspond to the probabilities for each slide at a given moment in the talk. The wider the stripe, the more Jabberwocky thinks it is likely the speaker is discussing that slide.

Looking at the speaker's narrative and comparing it to the slide contents, we can see that Jabberwocky usually switched to a new slide within five words of when the speaker started discussing it. The five words roughly correspond to two phrases. Looking at the probability distribution, we see that Jabberwocky was nearly always confident. Isolated spikes indicate places where Jabberwocky was distracted due to the speaker inadvertently saying a key phrase from a different slide. Even though none of these spikes were big enough to cause Jabberwocky to incorrectly change slides, they do suggest that it may be difficult to do large-scale free-form lectures without having unwanted slide changes.

5.3.3 Similar speech-based research

Jabberwocky's Bayesian method for skipping around in the slides has much in common with the methods used by Heckerman and Horvitz [42, 43] in the research that led to the Microsoft Office Assistant. They take typed free-text queries, extract key terms, and then use a Bayesian approach to determine which of the thousands of help topics will be most useful as an answer to the query. Like Jabberwocky, they do almost no grammatical analysis of the text, but unlike Jabberwocky, all of their probabilities were human-generated in an impressive knowledge-engineering effort.

Everett, Wauchope and Perez-Quinones [26] build natural language interfaces for virtual reality systems. A challenge they face in one of their virtual reality worlds is that nothing is labeled with names. As a result, different people will often use very different language in referring to the same objects. But, since the number of reasonable commands and queries is very limited, they are able to get good results by simply scanning for key words (i.e. words used in these commands) and then deciding what the person wants based on these key words and on what is around them in the virtual world. Like the Jabberwocky and Microsoft Office Assistant research, this work is able to use a shallow language understanding because it is able to severely limit the number of things that the user might say.

5.4 Learning what the speaker might say

There are two situations in which Jabberwocky fails to match the speaker's words to the correct slide: when the voice recognition drastically mishears what the speaker said and when the speaker uses too many synonyms in paraphrasing a point. In both cases, Jabberwocky's shallow understanding (of phonetic similarity and of synonymy)

renders it incapable of making the match.

When we examined the first situation, we immediately noticed that the ways that the speech recognizer misheard the speaker were somewhat consistent. With a phrase like "information systems", if the speech recognizer heard "and permission systems" once, it was likely to hear it that way much of the time. This means that if Jabberwocky listens while the speaker practices his presentation (even just a few times), it will be likely to encounter most of the recognition errors for that presentation. Better still, this makes it unnecessary for the speaker to go through the tedious process of training the voice recognition system, as Jabberwocky learns the few aspects of the speaker's talking that are needed for the presentation.

With the second situation, we were faced with a frustrating problem. Though we would like to use something like WordNet [62] to capture semantic similarities between words, in technical presentations, it will not find the important synonyms that are specific to a particular technical field. So, pending a better solution, we simply memorize phrases that seem potentially synonymous, so that they can be recognized later. As an added bonus, we found that this technique also deals well with canned slide transitions, where the speaker uses many phrases that are not a part of the slide contents.

In our methods for dealing with these situations where the basic techniques fail, we have consciously avoided forcing Jabberwocky to rely on a deep understanding of the slides' text or the speaker's words. To do otherwise would require a great deal of knowledge engineering–efforts that would be useful only in the narrow contexts of particular talk domains, or for the particular speech characteristics of a particular speaker. Instead, we rely on the speech recognizer and speaker to be reasonably

consistent so that we can simply match their words and phrases against their past words and phrases. To determine which words and phrases are worth remembering, Jabberwocky uses the same phrase extraction methods that it uses in analyzing the slide contents. In addition, because ViaVoice's vocabulary is much larger than the lexicon the Jabberwocky uses, we also extract large words that are not successfully tagged. This deals successfully with both mishearings and with obscure technical words.

5.4.1 How the learning takes place

Jabberwocky must do more than just identify what words and phrases are important to remember; it must also know what slides or slide points to associate with them. If it makes many bad associations, Jabberwocky is essentially learning how to do a bad job of running the slide presentation. Jabberwocky simply associates the phrases with the slides (and even slide points) that they were heard with. Because of this, while rehearsing with Jabberwocky, the speaker must make sure that he and Jabberwocky stay well synchronized.

When they do get out of synch, the speaker can inform Jabberwocky by issuing commands like "next slide" or "no, go back." The phrases immediately preceding the command are then associated either with the next slide (for the first command) or with the previous slide (for the second). A speaker may further aid Jabberwocky by telling which slide points he is lecturing from (as in "here is slide point one.") With speakers who reliably discuss every point in order on their slides, this level of training should help Jabberwocky to provide error-free control of their slides.

Figure 5.10 shows how Jabberwocky learns from the narration it hears. On the left are the three ways that Jabberwocky heard the speaker introduce the slide. The

The speaker said: You might wonder: "How does it know what I'm talking about?"

Jabberwocky heard (from the speech recognizer):	Phrases that were learned:
1) "You might wonder how does annoy them talking about"	do annoy, annoy, how do, wonder how, wonder might, might wonder, wonder, talk
2) "You wonder how does the northern talking about"	how do, wonder how, wonder, northern talk, talk .
3) "You wonder how is it now been talking of a"	wonder how, wonder, now talk, talk

Figure 5.10: Learning phrases in Jabberwocky. Across the top, what the speaker said. On the left, what Jabberwocky heard the three times the speaker spoke. On the right, the words and phrases that Jabberwocky learned.

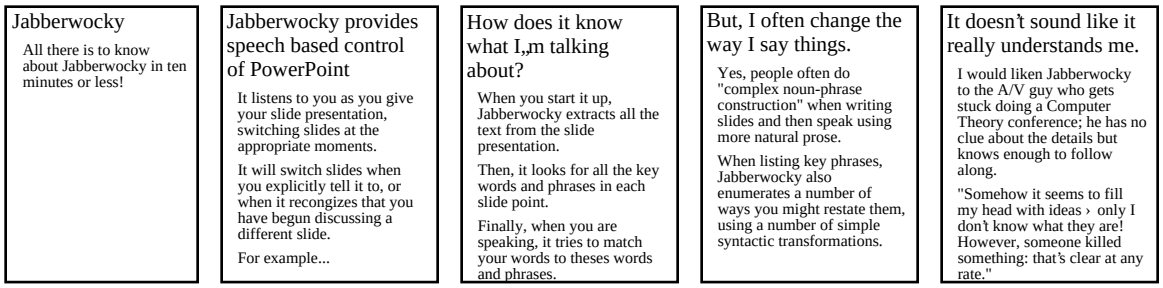


Figure 5.11: The five slides used in the second example

speaker added the phrase "you might wonder" as a transition into the slide and Jabberwocky learned several phrases involving the word "wonder." Now Jabberwocky can anticipate the speaker's transition. The speech recognizer had problems with the phrase "does it know what I'm" (monosyllabic words tend to cause the most difficulty). The phrases it learned as a result (involving "annoy" and "northern") will help it the next time the speech recognizer mishears the words in the same way.

5.4.2 An example of Jabberwocky learning new phrases

In this example, we look at how Jabberwocky learns new words and phrases as a speaker trains it by rehearsing a presentation three times. Here, the speaker used the same slides as in the previous example (the slides are pictured in Figure 5.11) and employed a somewhat consistent narrative in discussing them. The speaker guided Jabberwocky through the talk, using commands like "beginning the next slide" and "beginning the second point" so that Jabberwocky would be able to accurately associate the phrases with the right slide points.

Before the learning began, Jabberwocky had extracted 129 words and phrases from the slides, using the syntactic approaches discussed earlier. From the first rehearsal, Jabberwocky learned 43 new phrases. These consisted largely of phrases taken from speaker transitions, as well as a number of speech recognition errors. From the second rehearsal, Jabberwocky learned 24 new phrases, and rediscovered 14 phrases from the previous rehearsal. From the third rehearsal, Jabberwocky learned 18 new phrases, and rediscovered 23. The increase in rediscovered phrases is due to the speaker's consistency in making transitions between slides and the speech recognizer's consistency in mishearing certain phrases.

Figure 5.12 shows some interesting phrases that Jabberwocky learned for the fifth slide in the show, through the course of the rehearsals. The first two ("billion stance" and "clueless forgetting") are due to the speech recognizer mishearing the speaker's words. The third set of phrases is due to the speaker's setting up the quotation from "Through the Looking Glass."

New phrase: "billion stance"
Jabberwocky heard: "... doesn't delicate billion stance be no."
Speaker actually said: "... doesn't sound like it really understands me at all."

New phrase: "clueless forgetting"
Jabberwocky heard: "... theory conference is clueless forgetting what's been talked about."
Speaker actually said: "... theory conference is clueless regarding what's being talked about"

New phrases: "like alice", "lewis carroll", "look glass", "glass factory"
Jabberwocky heard: "... like alice from lewis carroll still looking glass factory..."
Speaker actually said: "... like Alice from Lewis Carroll's "Through the Looking Glass" after reading..."

Figure 5.12: Results from the second example: interesting phrases learned for the fifth slide

5.4.3 Ramifications on the Intelligent Classroom's design

In this chapter, we looked at how we extended the functionality of the Intelligent Classroom to deal with some particular problems that came up in providing a speech-based interface to a slide presentation. The phrase-matching component technically fit into the Classroom architecture, but the probabilistic approach and the learning were clearly outside, and prompted us to build a new system, rather than rework the existing one. The resulting system exhibited many of the cooperative characteristics that we strive for in the Classroom's design. Most significantly, it encouraged speakers to lecture in whatever presentation style they wished, trusting Jabberwocky to follow along. In fact, Jabberwocky is now a usable system (we have successfully given numerous short demonstrations in our lab), giving us the opportunity to actually invite other people to use it–to see whether it is also useful. Through these experiences we will learn ways in which to extend Jabberwocky to render it more useful to speakers lecturing from slides.

The phrase-matching component could be used in the Classroom for indexing

media other than slides. If a speaker provides brief synopses of all the video segments he may wish to include in his presentation, the Classroom would enable him to select one for playing by simply discussing it. If he was to say, "Play the video segment that discusses the evolution from Shakey at SRI to Chip at the University of Chicago" the Classroom could then decide which robotics video segment the speaker desired, cue the video tape to the appropriate segment, and play it. Similarly, the phrase-matching component could be adapted for looking for on-line resources during a presentation (using key words and phrases as the context for a web search tool like Watson [12]) or for locating appropriate technical papers (using a tool like Rosetta [10]).

It is a lot less clear how the probabilistic approach could be worked back into the Classroom. But, it is equally clear that, with a notion of likelihood, the Classroom could be made much more reliable, and as a result, much more cooperative. Section 4.5.1 (suggesting alternatives to the plan generalization method) outlined some ways that Classroom could select a representative plan (from its set of candidate explanations) more quickly. But, what we would really like is a notion of how likely it is, based on what we have observed, that the speaker is pursuing a given course of action. We essentially want a combination of some of the techniques suggested in Section 3.5.1 (dealing with sensory noise in monitoring processes) with the improved generalization methods. This is tricky, but would substantially improvement the Intelligent Classroom.

The Classroom was designed with a goal of facilitating learning; a significant portion of its inner operation is exposed through the Memory System interface, meaning that a learning component can observe ("snoop" on) the aspects of the Classroom's operation that it cares about, without disrupting anything. So, it is simple to compile

statistics about how long particular process steps take or what events are commonly observed or missed. Then, the learning component could find relationships between particular speakers, processes, or types of actions and what the Classroom observes. This information could be encoded into the process representation through adding new event handlers or perhaps through new extensions to the representation.

In summary, one of the triumphs of the Jabberwocky research is that we are able to use inaccurate speech recognition input to accurately control a presentation. The reason for this success is that, since Jabberwocky knows what it should hear, it is able to make strong assumptions when it hears things that are not exactly what it expects. This is why we can follow along with presentations that we do not quite understand–in observing a lecture at the rocket scientist convention, we can tell when it is the appropriate moment to change the slides. In all of the tasks in the Intelligent Classroom, we use our task knowledge to make less smarts go further.

Chapter 6

Related work

The research detailed in this dissertation draws from several research areas. This chapter looks at other research that is relevant to and/or influential in the design of the Intelligent Classroom. Each section examines one of the related research areas, discussing particular projects and researchers that best typify that area.

6.1 Intelligent environments

In the 1998 AAAI Spring Symposium on Intelligent Environments [5], Michael Coen defined Intelligent Environments as computational spaces that provide "different forms of natural, multimodal human-computer interaction during what is traditionally considered non-computational activity." In the projects discussed during the symposium, interaction was facilitated using cameras, motion detectors, pressure sensors, microphones, and even specially instrumented household appliances. Interaction with these systems ranged from strongly task-oriented to frustratingly naive. In this section, I discuss some of the Intelligent Environments research that is most similar to the Intelligent Classroom.

6.1.1 MIT: KidsRoom, Intelligent Room and Hal

One of the earliest Intelligent Environments, MIT's KidsRoom [8, 9], is an interactive entertainment space, which immerses a group of children in a fanciful adventure. Starting in an ordinary-looking bedroom, the children are taken on a 10-15 minute journey, rafting through river rapids and dancing with monsters. The KidsRoom dictates the storyline and uses the context of what is going on in the story to tune its sensing systems, allowing great interaction using relatively unsophisticated computer vision and audio analysis techniques. At every point in the story, the KidsRoom knows what is reasonable for the children to do and uses this knowledge to aid its action recognition algorithms. For example, when the children are steering through the rapids, the KidsRoom uses a simple motion detection algorithm to determine which side of the raft the children are paddling on, and uses background-subtraction to notice if somebody falls off the raft. As a result, the sensing tasks are rendered sufficiently reliable to successfully immerse the children in the story, unaware of and not caring about what is going on behind the scenes.

While the KidsRoom is a model of how task context can be used to aid sensing, it is not a model of robust user interaction; the children cannot change the story–only stall it momentarily. The Intelligent Room is an example of an Intelligent Environment that is directly controlled by its occupants [20]. It is a conference room featuring three display screens, numerous wireless microphones and a dozen cameras (to catch all the action). The people in the room control it through spoken commands and gestures. To deal with the difficulty of controlling all the components, the researchers developed Scatterbrain [19], which is used to coordinate the fifty intercommunicating software agents that control the room. The room uses specialized vision systems for

person tracking, recognizing pointing gestures and recognizing gestures on top of the main table. The low-level techniques employed in this project are similar to those used in the Classroom, but the interaction in the Intelligent Room is limited to a set of commands, ignoring much of the useful context of the tasks it is to be used for. For example, when serving as a rescue command post, interaction would be facilitated if the room had a notion of all the steps required to set up an evacuation plan.

The highly specialized vision techniques employed in the Intelligent Room are taken even further in Hal [21], an Intelligent Environment/Office. For example, when people are moving about in the room, rather than tracking them, Hal will simply wait until they arrive; it is easier to notice when someone sits down on a couch than to follow him through a cluttered space as he moves there. As a result, the office is filled with numerous extremely specialized visual sensors (e.g. a "someone passed through the door" sensor) that are very simple and quite reliable. However, Hal suffers from having even less of a notion of "task" than the Intelligent Room. Hal's most important behaviors surround its role as an alarm clock for napping graduate students.

6.1.2 Xerox Parc: VizSpace Project

A similar research project, Xerox Parc's VizSpace [60], serves as a test-bed for device-less multi-modal user interfaces. Through the combination of speech and gestures, its users can manipulate graphical objects on its display: adding and removing objects, moving them, and resizing them. The commands that users use to interact with VizSpace generally involve a speech command paired with a gesture (e.g. "add a sphere (the user points to a location on the screen) here"). Though the commands themselves are atomic, the system must coordinate the sensory results from the computer

vision and speech recognition systems, each with their own associated sensor delay. Unlike any of the previously described systems, VizSpace does not have a tangible task that it has been designed for; the researchers hope that this technology will prove useful in designing future user interfaces for new kinds of applications.

6.1.3 Georgia Tech: eClass (formally Classroom 2000)

In name, the Intelligent Classroom's closest kin is Georgia Tech's eClass (formally the Classroom 2000 [11, 1]). It is an Intelligent Environment set in a college classroom domain, with microphones and cameras to observe the lectures. But, rather than serving as an audio/visual assistant to help the lecturer, it serves as a note-taker to help the students. The eClass stores the contents of the marker boards and the slides that were shown, as well as the video and audio of the lecture. These resources are compiled into a web site that indexes them using an annotated timeline showing key events of the lecture. The eClass is also unique in that it is fully deployed in an actual classroom and has been in use for a few years. It is apparently very popular with many students who claim that they are better able to participate in classroom discussions since they are freed from madly copying everything that the lecturer writes on the board. Unlike the Intelligent Classroom, the eClass does not try to follow along with what the lecturer does; instead it constantly collects all the resources, letting the students sort it out for themselves later.

6.2 Plan recognition in service of a task

All plan recognizers must have some bias; otherwise they cannot infer anything [57]. For example, a completely unbiased plan recognizer, faced with a sequence of ten

actions to explain, would have to consider the possibility that each of the ten actions is a part of different plan. Such a system would do a bad job of explaining an agent's actions and an even worse job at predicting what it might do next. As a result, all plan recognition researchers add some sort of bias to their systems. Kautz' "generalized" plan recognizer has the rather general bias of preferring explanations that include fewer plans [51]. But, when plan recognition is used in service of a particular task, the recognizer can be strongly biased to help it in its task. In the Intelligent Classroom, these sorts of biases were suggested throughout Section 4.5. In this section, we look at several other research projects that rely heavily on plan recognition in performing specific tasks.

6.2.1 ISI: Artificial fighter pilots

At the Information Sciences Institute of the University of Southern California (ISI) there are a number of research projects in support of building artificial fighter (or helicopter) pilots for a simulated air-combat environment. [75] describes RESC, an approach for recognizing and reacting to the perceived plans of other agents. One capability that the fighter pilots need is to be able to infer actions that cannot be sensed (e.g. the firing of an air-to-air missile) but that need to be acted upon. Although the actual firing of the missile cannot be sensed, some of the other actions that comprise a missile-firing plan can be. Since the cost of failing to evade an incoming missile is so high, the system is strongly biased to recognize when this and other threats are possible. Like the Intelligent Classroom (and unlike most general plan recognition systems), the artificial fighter pilots in this research nearly always have to initiate their actions before they are sure of their plan recognition results. This, in combination with the system's trigger-happy disposition, frequently forces the system to

abandon its planned course of action. Due to the structure of the plans used in this domain, RESC can utilize a more-efficient "single-state backtracking" that utilizes the previous plan recognition results when it is forced to revise its understanding. The way that RESC combines plan recognition and plan execution is quite similar to how the Intelligent Classroom employs them.

The fighter pilots need to be aware of the plans of both their enemies and their allies. In particular, when pilots are flying in formation, they need to reason about their team's mission, their role in that mission, and what they can expect the other pilots around them to do in pursuing the mission [74, 50]. For instance, without a notion of team behavior during reconnaissance missions, the researchers found that an artificial pilot would often miss a signal or fail to observe a landmark and end up wandering off by itself. RESC-team was developed to extend RESC to reason about teams and team-member's roles, recognizing situations where the team is breaking apart. The plans in RESC-team describe how the various roles in the plan fit together and what actions should be observed. When a pilot observes that other agents are not acting in agreement with its expectations, the pilot signals that something is wrong, prompting the team leader to break radio-silence to get the team-members back together.

6.2.2 University of Michigan: Netrek player

Researchers at the University of Michigan implemented agents to plan the net game Netrek, in which teams of space ships fly around the galaxy, trying to take over more planets than their opponents. The game combines strategy and hand-eye coordination as players try to out-think one-another in planning campaigns through the galaxy and out-fight one-another in one-on-one space combat. The first generation agents

were developed using the Procedural Reasoning System (PRS), described later in this chapter. These agents performed very poorly; though they were formidable dogfighters, they were also extremely predictable and would stubbornly refuse to abandon a mission, even if it was rendered doomed or counterproductive. Human players could easily recognize the agents' deficiencies and exploit them.

Then, the agent plans were transformed into probabilistic belief networks for doing plan recognition using ASPRN (Automated Synthesis of Plan Recognition Networks) [44], and the resulting plan recognizers incorporated into the design of the agents [45]. The resulting agents, when pitted against the original ones, were then able to recognize what their opponents were doing. As should be expected, the plan-recognizing agents had no difficulty demolishing the others. Unfortunately, these researchers did not publish descriptions of how the plan recognition results were used to thwart the opponent's efforts, so it is not possible to compare these techniques with those used in the Classroom.

6.2.3 Microsoft: Office Assistant

While the last couple of research projects have been aimed at recognizing the plans of opponents in order to foil them, many researchers have, like the Classroom, employed plan recognition in order to assist human users in their plans. In electronic domains, most research strives to strike a delicate balance: they wish to provide valuable assistance and yet not annoy their users [68]. At Microsoft, the researchers behind the most demonized user interface agents today have done some of the most compelling research on how to forge a good compromise. The Lumiere Project [43], which was eventually watered down to provide the basis for the Office Assistant, employs Bayesian user modeling to try to infer the goals and needs of people as they

use Microsoft Office. The basic idea is that, as it observes the user's actions, Lumiere builds models of what they are trying to do (i.e. their "goals"), what information might be useful to them (i.e. their "needs"), and also what their strengths and weaknesses are in using the software (i.e. their "competences"). They developed a language for representing high-level actions as patterns of low-level actions such as mouse clicks and keystrokes. The high-level actions that are recognized are then used as inputs to a large Bayesian network that computes probabilities for: (1) the various activities the person might be engaged in, (2) what information might be useful in that task, and (3) whether the person desires any assistance. In this research, the plan recognizer is essentially a simple pattern matcher; the most interesting part of the research is how the plan recognition results also determine when and how to assist the user. The implementers of the Office Assistant did not implement much of the functionality developed in the Lumiere Project, which perhaps explains the Assistant's many shortcomings. The pattern recognition used in this work to recognize user plans is not as flexible as that used in the Classroom, but the combination of user modeling and goal inference developed for the Lumiere Project would be a welcome addition to any interactive agent, including the Intelligent Classroom.

In Microsoft Office's help system, a Bayesian approach is used to determine which help topics might be relevant to a users free-text query [42]. The system does not do any deep Natural Language understanding of the query. Instead, it looks to see which keywords (from a set of several thousand, hand-selected words and phrases) the user included in their question. Both the presence and the absence of particular keywords is used as evidence in the Bayesian network, which produces an estimate as to how likely it is that the user is interested in each help topic. The probabilities

needed by the network–probably hundreds of thousands–were human-estimated in a massive knowledge engineering effort. The help system does not consider the user's recent actions and tasks when it chooses help topics for his query. Its approach (and unreliable results) is similar to Jabberwocky with a large unstructured presentation. But where Jabberwocky is choosing among twenty or thirty slides, the help system has to choose among thousands of help topics.

6.2.4 Medicine

TraumaTIQ [40] is a plan recognition system that observes and critiques trauma patients' care in a hospital's emergency room. The system looks at the primary physician's various diagnostic and treatment actions (entered into a computer by hand to document the patient's treatment) and tries to infer the reasons behind them. Then, when the system disagrees with the physician's actions (e.g. if the doctor leaves out important actions, does unnecessary tests, uses an unpreferred procedure or does actions in a bad order), it will tell the physician. This system is an excellent example of how plan recognition can be made useful by using it in a well-understood domain. Emergency room physicians are given very specific training as to how they are to deal with the vast majority of ER patients, and their treatment plans are easily represented and recognized. The TraumAID system [76] previously looked at what actions to take to best diagnose and treat trauma patients.

6.2.5 Collagen

Researchers at Mitsubishi Electric Research Laboratories (MERL) have developed Collagen, a framework for developing intelligent user interfaces based upon the principles underlying human collaboration [69]. Collagen has been used to implement

interfaces for making plane reservations, operating VCRs and thermostats, developing graphical interfaces, and teaching student users how to operate a gas turbine engine and generator configuration [70]. The framework derives its strength by separating the general collaboration principles from the task-specific ones. The general collaboration principles include the *types* of task knowledge an agent ought to be reason about and communicate. These include things like: "where are we in this task?", "what should we do next?", and "why did we just do that?" While the agent is collaborating in a task, it tries to maintain an accurate discourse state: a model of the status of the task. A Collagen agent will use its actions, the user's actions and plan recognition techniques to update the discourse state during a task. This discourse state uses a task representation that is functionally similar to the Intelligent Classroom's processes (i.e. agent actions advance an activity's state). It is not clear what would be needed to extend Collagen into a physically interactive system like the Classroom, but the collaborative reasoning used in Collagen would be quite useful, especially in diagnosing what went wrong when the Classroom took a strange action.

6.3 General plan recognition

Any plan recognizing system, regardless of its plan recognition bias, must employ some basic algorithm (into which the bias is then added). This section looks at a few researchers whose work has been the most influential in general plan recognition.

6.3.1 Wilensky

PAM is one of the first plan recognition systems. (See [77] for a detailed description of its implementation.) This program takes conceptual dependency representation [72] of

short story fragments (less than ten sentences) and attempts to build a representation that explains them. The system can then use this representation to answer questions about what happened in the story and to retell the story from the perspective of any character in the story. In order to understand a story, a system must understand the character's motivations for their actions; this boils down to simple plan recognition. PAM works incrementally; for each new sentence, the program either recognizes how it fits in with its current hypotheses of the characters' plans and goals or, failing that, develops new hypotheses to explain the sentence.

One element of story understanding that PAM addresses is that of recognizing goal relationships. This is important because good stories are built around sets of interacting plans and goals. Four goal relationships the PAM recognizes are: (1) where numerous recurrences of a goal can be dealt what at one time, (2) where the goals of a single agent are conflicting, (3) where the goals of different agents are in competition with each other, and (4) where the goals of different agents are in accordance. Recognizing these relationships gives a plan recognition system a great deal of explaining and predicting power.

Though the plan-recognizing component of PAM is situated in the task of understanding stories, this domain does not add the strong sort of bias discussed in the previous section; ultimately, the actors in the stories can be expected to do almost anything. The incremental hypothesis-building process utilized in PAM is very similar to the candidate proposing and refining techniques employed in the Classroom. But the two systems employ the results in very different ways.

6.3.2 Kautz

Kautz, in [51], describes a technique for performing plan recognition using propositional logic; once all of the logical axioms are set up, plan recognition is accomplished using a general-purpose theorem prover. To accomplish the most general plan recognition possible, pure logic is great; any system biases have to be explicitly added (as axioms). However, from an implementational standpoint, it is not clear that such systems can be efficient and it is not known how to easily add biases to the recognizers. (The system requires the use of circumscription and Kautz is not aware of any universally applicable methods for automating circumscription.)

Another major contribution of this research is in its use of a hierarchy of plans in plan recognition. When the system does not have enough information to determine the precise plan, it reasons about the *class* of plan it is recognizing. For instance, in observing someone prepare a meal, the system may fail to identify the particular kind of pasta or sauce the chef uses; then the system will recognize that the chef is cooking a pasta dish. The Classroom employs a plan hierarchy inspired by this work, allowing it to make predictions and act on the (limited) available information in situations where it cannot afford to wait.

6.3.3 Charniak

While Kautz transformed plan recognition into theorem proving, in [18], Charniak views it as "a problem of inference under conditions of uncertainty", justifying the introduction of Bayesian (probabilistic) reasoning. Ideally, the plan recognition process would be dealt with using a single, massive Bayesian network. But, due to the many problems of dealing with large Bayesian networks, the task of plan recognition

is broken into two subtasks. The first is to retrieve a set of candidate explanations for a set of actions, and the second is to assemble a plan recognition Bayesian network for those explanations, allowing the system to compute which explanation is most probable, given the observed actions.

The system decides which plans are candidates using a marker-passing scheme: as new evidence is presented, markers are sent out from the relevant nodes in a semantic network that holds plans, actions and objects in the world and links that describe the different relationships between them. Each intersection the system finds corresponds to a potential candidate explanation for the set of observations.

Each candidate explanation is added into a Bayesian network that is constructed from scratch each time the plan recognition process starts. New nodes and connections are added to the network based on a well-defined set of construction rules. Once the network is completed, the plan probabilities are computed and the most probable plan, based on all the evidence, is adopted.

The probabilistic approach often yields very intuitive results (e.g. particularly unlikely plans are not seriously considered until sufficient evidence supports them). However, probabilistic systems require vast quantities of probabilities in order to operate and when a designer employs unprincipled guessing to get the needed numbers, the system's behavior often becomes frustratingly unintuitive again.

6.3.4 Konolige and Pollack

In [53], plan recognition is viewed as ascribing beliefs and intentions to other agents. Plan recognition is accomplished though the interaction of two types of information: local cues derived from the data (observed actions and ascribed beliefs of the actor) and the coherence of local information in a global structure. The processes for dealing

with these information types are bottom-up (from local cues) and top-down (from global structure.)

The paper discusses a number of inferences a system should make based on specific kinds of observations. These are the general plan recognition biases of the system. Here are several of the inferences making up the system's bias. (1) If an agent expects something to be true in the future, it will not execute a plan to achieve it. (2) If the observer knows something, then any other actor will know it also. (3) If an agent tries to achieve something, then it will expect it to become true. (4) If an agent believes that a proposition is an effect of taking an action and intends to take that action and has a goal to achieve that proposition, then it intends to do the action to achieve the proposition.

The paper also proposes a technique for deciding what beliefs and intentions to ascribe. This is done using the process of argumentation, where support for ascriptions is compiled and conflicts between ascriptions are detected, and finally decisions are made based upon rules for accepting and defeating arguments.

The observing agent described in the paper is an errand robot, which needs to reason about the taskmaster's goals in interpreting ambiguous or contradictory commands. The Classroom, which relies more on the question of "what" than the question of "why", does not need to make use of these sorts of inferences. But, an approach such as this one would be necessary for an assistive agent that would perform independent tasks for its user.

6.4 Other related research

Finally, there are a number of other projects and research areas that are also of interest with respect to the Intelligent Classroom. These projects address some of the same issues as the Intelligent Classroom (e.g. performing camerawork, using the results of speech recognition, and executing and monitoring plans that do not abide by the simplifying assumptions of classical planning).

6.4.1 MIT: SmartCams

A SmartCam is a computer controlled television camera that uses computer vision to guide its movements. These cameras serve as an important technology in developing a fully automated "Intelligent Studio" for filming TV shows [65]. In designing the SmartCams, the researchers strove to make the automated cameras look at the problem from the perspective of a human camera operator, asking questions like, "what does the director say?" and "what details are important for this particular shot?" The computer vision system behind the SmartCams uses two levels of representation: a high-level "approximate world model" and a low-level "view representation". The high-level representation is three-dimensional (made up of cylinders and ellipsoids) and is used as a starting point for developing the low-level two-dimensional representation.

Like the KidsRoom project, a major strength of the SmartCams research is in its ability to apply contextual information to render the computer vision tractable and reliable [66]. The SmartCams are demonstrated in a cooking show domain. The contextual information is expressed in a detailed script of the show, which includes the actions of the chef (e.g. "chef pounds chicken with a wooden mallet") and the

desired camera shots (e.g. "close-up chef"). Based on the action and desired camera shot, the SmartCam determines the appropriate vision technique to employ–often it is sufficient to use simple frame differencing, with the high-level representation providing a starting point.

However, the SmartCams could not be employed in the Classroom because they are very early prototypes that lack crucial functionality. First, the cameras' motions are all simulated–the vision techniques would not work with moving cameras. There are three fixed cameras (front, side and overhead) and the SmartCams "frame" a shot by selecting a portion of a camera frame. Second, the show's script is annotated by hand and, as a result, the system works off-line. A human assistant must indicate which camera frames correspond to each action in the script. Regardless, this research is a great piece of vision work that works an effective compromise between reconstructionist and purely purposive approaches.

6.4.2 Using Speech Recognition

At Rochester, the TRAINS [3] and TRIPS [27] projects demonstrate powerful ways to integrate speech recognition input into a planful computer assistant. The primary goal for each of these projects is "to demonstrate that is feasible to construct robust spoken natural dialogue systems." In natural dialog, the user must not be limited to a fixed set of utterances; so the speech recognizer cannot use a command grammar to aid it. The systems perform chart parsing on the noisy speech recognition input (approximately 70% accuracy), refining the input during each step of the understanding process.

The TRAINS system helps a user find a route between cities, trying to find efficient paths between pairs of cities. The user might start an interaction with TRAINS by

saying, "Okay. Let's take a train from Detroit to Washington." Given an utterance like this, TRAINS first performs statistical error correction, using training results from a corpus of previously transcribed dialogs with the system. Second, the corrected utterance is sent to a speech act parser, which looks for all the plausible speech acts in it. Finally, TRAINS interprets the speech acts in the context of what the system and user are doing. This process allows the system to recover from speech recognition errors and also to understand the sorts of speech fragments that are common in human dialog.

The TRIPS system (The Rochester Interactive Planning System) applies similar techniques to collaborating with a human user to help manage rescue resources in a hurricane crisis situation. Together, TRIPS and its user construct plans for evacuating people to a safe city on a fictional island. There are multiple modes of transportation and roads might dynamically be washed out. The plans and richness of dialog needed in TRIPS are much richer than those in TRAINS, making TRIPS a good research successor to the TRAINS system.

At the Naval Research Laboratory there are a number of projects that involve providing spoken natural language interfaces to virtual reality systems [26]. A challenge the researchers face in one of their virtual reality worlds is that nothing is labeled with names. So, different people often use very different language in referring to the same objects. But, since the number of reasonable commands and queries is very limited, they are able to get good results by simply scanning for key words and then deciding what the person wants based on these key words and on what is around the person in the virtual world. In another virtual reality world, there is less of a naming problem and they focus on stronger NL parsing approaches, using knowledge about

the current situation to filter nonsensical parses.

6.4.3 Execution and Monitoring

IPEM (Integrated Planning, Execution and Monitoring) is a nonlinear planner (in the NOAH [71] and TWEAK [17] tradition) with execution monitoring and replanning capabilities [4]. The execution module has access to the complete plan structure (with all the protected links, etc.) and notices when unexpected events either remove a plan step's preconditions or make it redundant (by achieving its post-conditions). In such situations, it will replan to deal with the new situation. Also, the system provides a simple way of incorporating sensing actions into its plans: it uses special variables to store the sensing results for use in later steps in the plan. For instance, in a plan for a robot to go into the next room and pick up a wrench, the plan does not need to know the unique identity of the wrench. Instead, the plan will include a step to locate the wrench, once it has entered the room, and then have a step to pick up the wrench that it found there.

EXCALIBUR is a system in which qualitative process theory is used with a typical state-based planner (Nonlin [71]) to better reason about domains where there are continuous processes [24]. Qualitative process theory allows a reasoner to deal correctly with the indirect effects of actions–the action of opening a faucet leads to the condition of the faucet being open, but a state-based planner will have trouble predicting whether any water will flow and how long it might last. The fundamental idea is that the actions of an agent often cause a process to start. In many cases, it is the process that causes a desired condition, not the action itself. The process examples given in this research are of heat-flow, fluid-flow, boiling (to produce steam) and motion.

An important component of EXCALIBUR is its process reasoner, which creates a process tree from its plans to reflect possible behaviors of the system. It does this by causing branches to be formed for each possible termination of a process (for a fluid flow between two containers, this might include balance of pressure, one container overfilling, or one container emptying). This structure allows the system to see what might happen through the execution of its plan and help it identify possible problems (like explosions or overflows). The process reasoner also uses sensory input to synchronize the internal world model and modify the process tree as necessary. Nodes in the process tree are essentially states and they hold objects relevant to the state, processes active in the state and the influences the processes cause in the state. The Intelligent Classroom applies this process notion to agent plans; the addition of an EXCALIBUR-style process reasoner might aid the Classroom in reasoning about possible outcomes of user actions.

In [25], the process theory techniques used in EXCALIBUR are expanded to deal with more complicated kinds of processes: such as the process of the functioning of a clothes washing machine. This research is in developing an agent who can learn about the details of a process (e.g. what sounds the machine makes, what buttons light up, when these things happen, and how long the different phases take). The key contribution of this research is the idea that plans and processes can be represented in the same way: a sequence of steps that are linked together in a potentially complicated way. Based on this idea, the research describes how an execution system can start with a skeleton plan and fill in details through repeated execution.

PRS (Procedural Reasoning System) attempts to produce the kind of behavior expected of a rational agent, by explicitly representing the psychological attitudes

of beliefs, desires and intentions. (Agents that are endowed with these attitudes are often referred to as BDI agents.) In [39], PRS is used in the navigation portion of the "Flakey the robot in a space station" project. The BDI components in PRS are implemented as follows: beliefs are a system data base of facts, desires are a set of goals or, more accurately, desired behaviors of the system, and intentions are a process stack of active Knowledge Areas. Knowledge Areas (KAs) describe certain sequences of actions to use to achieve goals or respond to particular situations (or combinations of goals and situations). The KAs facilitate both planful and reactive behavior. Also, since the BDI components are explicitly represented, metalevel-KAs can reason about and alter the system's beliefs, desires and intentions. PRS is a popular reasoning system due to its intuitive way of representing an agent's volition, and the ease of shifting between planful and reactive behaviors based on context.

BURIDAN [55] is "a sound, complete, and fully implemented least-commitment planner whose underlying semantics is probabilistic." The system uses a standard nonlinear planning algorithm (most similar to UCPOP [64]), but the uncertainty about the initial world state is captured in a probability distribution over states, and actions have probabilistic outcomes (e.g. for action `pick-up`, `[gripper-dry, 0.95, holding-block]` indicates that if you try to pick up a block when your gripper is dry, you have a 95% chance of ending up holding the block). With threatened links, the planning algorithm can promote or demote (as in UCPOP) or confront (i.e. plan, assuming that the threat will not happen.) As far as plan correctness, BURIDAN will produce plans with a desired level of probability of success.

Chapter 7

Conclusion

This dissertation discusses techniques for building systems that share the effort of doing tasks with their users. One important aspect of such systems is that they must understand–to a limited degree–what their users are doing and how they are expected to interact. I use the term "competent assistant" to refer to a system that has a simple understanding of the sorts of tasks it will be employed in and uses this understanding to cooperate with its users. These software and hardware systems can aid a user in his work because they understand what that work is. To be successful, such systems must sense a user's actions, recognize what task he is performing and then, at the appropriate moments and in the appropriate ways, assist him. Much current research in this area has focused on software agents like the various Microsoft Assistants [43], travel assistants [69], or other information agents [61]. These agents tend to promote successful interaction to the degree that they are targeted at particular tasks; an agent trying to assist in the domain of all word processing tasks is faced with a nearly intractable assistance problem, while one that just tries to help a traveler put together an itinerary can cooperate with relative ease.

With all this in mind, this dissertation introduces the Intelligent Classroom: an

automated lecture facility that serves as its own audio/visual coordinator. The Classroom has an understanding of the sorts of tasks that its user (a speaker) is likely to undertake as a part of his presentation. And, as the speaker lectures, the Classroom recognizes (through plan recognition) what tasks he is performing and uses its task understanding to assist, providing behind-the-scenes support that makes the speaker's presentation go smoothly. In the current incarnation of the Classroom, this support involves setting up and controlling the various audio/visual components and producing a videotape of the presentation.

The research described in this dissertation details the entire process of taking low-level sensor data, determining what high-level task the speaker is performing, deciding how to assist, and setting the Classroom's actuators to accomplish the cooperation. For example, based on the speaker performing a particular gesture, the Classroom may decide that the speaker wishes to pause the videotape that is playing, prompting the Classroom to control the VCR in the way the speaker desires.

In this chapter, we address some potential complaints about this research that have not been discussed elsewhere in the dissertation. Then, we look at some of the philosophical commitments of this research: project-independent goals that have guided this research. Finally, we point out a couple more issues that offer interesting future research directions, for this project as well any other research dealing with human-user collaboration.

7.1 Some potential complaints

The Intelligent Classroom takes a strongly representational approach to interacting with the speaker. It explicitly represents the plans and processes that the speaker

might use in doing a presentation, using these representations in recognizing what the speaker is doing and in determining how to act in response. Some researchers might question whether so much reasoning is desirable in competent assistants, asking whether a technique that is more computationally efficient would be just as good or even better. Others might question whether the Classroom domain warrants this much reasoning. In this section, I seek to provide answers to these potential complaints.

7.1.1 Efficiency: why not just use a rule-based system?

While the Intelligent Classroom is performing plan recognition, a single speaker action can cause a great deal of Classroom cogitation. The Classroom must determine whether that action is consistent with each of its candidate explanations, and, if any of the explanations are eliminated, the Classroom must also revise its representative process (i.e. the explanation that it is currently acting on). In response, someone may ask, "Wouldn't it be possible to get the same interactive results through a much more computationally efficient process?"

Artificial Intelligence researchers have devised plan-like representations that allow an agent to take goal-directed action in the face of uncertainty regarding the outcomes of its actions and regarding what actions other agents might take. For instance, Universal Plans [73] are tree-like plan structures where branches in the tree correspond to moments in the execution of the plan where, at planning time, the planner cannot be certain what will happen next. At the other extreme, there are flat rule-based systems that use stimulus-response pairs of the form, "if you are in this situation and this action occurs, take this action." In such systems, the behavior of a system is distributed among a set of hundreds or even thousands of rules. Soar [56] is an example

of an extremely developed rule-based system with many optimizations designed to use the agent's current context to determine which rules to consider.

Despite the fact that these approaches were developed to produce goal-directed agent behavior, it is not difficult to use them to produce the sort of behavior desired in the Classroom: taking cooperative action while determining precisely what the speaker is doing. In a plan recognizing Universal Plan, the root of the plan/tree is the situation where the Classroom does not know what the speaker is doing: it has just started trying to recognize the speaker's plan. The leaves of the tree are the various results of the plan recognition process. In the internal nodes, the Classroom is taking action and refining its understanding of what the speaker is doing, based on the actions he takes. (In such a system, the Classroom's understanding of what the speaker is doing is encoded in the Classroom's current location in the plan/tree. In a rule-based system, plan recognition is accomplished through manipulating the system's internal state: an action changes memory propositions to reflect how it affected the Classroom's understanding of the speaker's activity.

It would not be difficult to transform the Classroom's plan and process representations into either of these other representations; the various states of the Classroom's recognition process map into nodes of the plan/tree or situations in the rules, the actions the Classroom might sense map into transitions leading out tree branches or actions in the rules. It seems likely, however, that the resulting systems would be much too large to deal with. There are simply too many possible states the Classroom might be in and too many possible actions that the speaker might take.

Furthermore, even if the resulting systems were manageable, they would have the undesirable characteristic of being difficult to change to suit the needs of particular

speakers. For instance, if a speaker decides that he never wants the Classroom to advance slides when he makes a particular slide-advancing gesture, the tree-based or rule-based system would have to be completely rebuilt to incorporate his wishes. This seemingly minor change in behavior requires tens or even hundreds of changes throughout the system. In the Classroom's implementation, this change would simply require a one-line alteration in the context of the definition of the process that deals with the gesture to specify that the speaker does not ever perform it. Adding behaviors is similarly difficult in the tree-based or rule-based systems and easy in the Classroom.

This behavioral flexibility is crucial in an interactive system like the Intelligent Classroom; it is only as people interact with the system that they realize which behaviors they dislike and which ones they would like to add. In tree-based and rule-based representations, the process of determining what a speaker is doing is explicitly encoded and the interactions between competing explanations are fully spelled out. The Classroom's representations and algorithms take care of the tedious interactions automatically, allowing researchers to manipulate the systems behavior at a more natural level.

7.1.2 Overkill: do you really need this much reasoning for this domain?

It seems plausible that a satisfactory automated classroom, much like the Intelligent Classroom, could be implemented using a much simpler approach. Even if someone is convinced by the last section's argument that the type of plan recognizing the Classroom does requires the sort of reasoning described in this dissertation, they still might wonder if the domain of automated classrooms warrants this much reasoning.

They might ask, "In a domain that simply requires the system to aim cameras and run A/V equipment, do we really need a full-fledged plan recognizer?"

Perhaps surprisingly, I would respond that, for an actual implementation of such a classroom, we would almost certainly not need to do the sort of plan recognition described in this dissertation. Instead, once the desired behavior of the Classroom was well understood, it might be easy to implement a version of the Classroom for that desired behavior. For the Classroom domain, such a system could probably be implemented to run substantially faster and using much less memory than the Classroom's current implementation. Note, however, that such a system would not be flexible enough to support the sort of personalization discussed later in this chapter.

The issue, however, is how we can determine what the desired behavior is. Through introspection about the activities involved in giving a presentation, we can reason about much of the fundamental behavior we would want in the Classroom. But, it is only through experimentation with actual speakers giving actual presentations that we can hope to discover the more subtle behaviors needed to make such a system a success. So, as a prototype, the flexibility afforded by the Classroom's reasoning ability is critical in discovering what behaviors are crucial and which are just bothersome.

7.2 Philosophical commitments

In designing the Intelligent Classroom, there were a couple of principles the pervaded our decision-making process and that proved beneficial in the prototype. These are general philosophical commitments that are applicable outside the realm of competent assistants. They serve to guide designers in producing a wide range of interactive

systems.

7.2.1 The "InfoLab theory of interaction"

Many of the research projects in the Northwestern Intelligent Information Laboratory (a.k.a. the InfoLab) strive to provide information and assistance to human users as these users go about their tasks. The Watson system [12] finds useful web pages, news stories, images, video segments, and items for sale, based on the user's browsing or word processing behavior. The XLibris system [22] automatically compiles a set of resources for a student who checks out a library book; it generates a web page that holds a summary of the book, related books, relevant on-line journal articles, and even a link to order the Cliff's Notes version of the book.

The theory of interaction behind these projects is that the user goes about his tasks (e.g. browsing the web or going to the library) and the system looks for opportunities to provide assistance. The user should not be required to ask the system for assistance; the system should simply work in the background and present its results in a way that does not distract the user from his task and yet, when appropriate, lets the user know that assistance is available. For instance, Watson attaches a small status panel to the side of the browser or word processor window, signifying when it is looking for useful resources and, once it finds some, how good it thinks its results are. XLibris sends the library patron an e-mail message with the URL of the generated web page, which the student can then view at his leisure.

An underlying idea behind this is that if you require a user to ask for assistance, he usually won't. Furthermore, even if he wishes to ask for assistance, he often will not know how to ask for the sort of assistance that he desires. In the Intelligent

Classroom, the speaker should just go about his presentation, trusting the Classroom to take the appropriate actions at the appropriate times, and commanding the Classroom only when absolutely necessary. Similarly, with Jabberwocky, the speaker should just lecture, letting Jabberwocky keep track of the slides. The user goes about his task, while the system observes him, providing information and assistance when appropriate.

7.2.2 The advantage of situatedness

When people use software systems, it is generally in service of some task. Knowledge of this task is required to understand their actions and anticipate their needs. The software is situated in its task domain and this domain provides two contexts that drive the interaction with the user. The first is the general context of the task (or set of tasks) for which the system is constructed. The second is the specific context of how the user is interacting with the system at the moment: the particular task the user is performing and the relevant details of how the user is performing it.

In Watson, there are two tasks that the user might be performing: looking on the Internet for information, or writing a paper. The specific context is the general content of what the user is currently looking at and his most recent actions. Watson uses the content and structure of the user's current document (in the web browser or word processor) to extract about twenty words representing the document's content area. Because Watson is used to find useful web resources using web search tools (general task knowledge), these twenty words are a sufficient summary of the user's current interest. The user's actions are used to determine when it is appropriate to use this summary to search for relevant information on the Internet. For example, if the user opens a new document, Watson will begin a new search. Similarly, if the user

adds a figure caption to his word processing document, Watson will use the caption and its document summary to locate possible images that could be used in the figure. Finally, if the user types a short query into Watson's "query in context" box, Watson will combine the query with its document summary to locate more targeted search results. The system interprets the user's actions in the context of the current situation and so is able to make much stronger inferences about what to do in response.

In Gargoyle (i.e. the computer vision system used in the Classroom and described in Section 2.2), the context of what the speaker is doing is used to determine what techniques to use to sense where the speaker is. The speaker does not directly use Gargoyle, so in discussing how it is situated, it is important to look at it as a part of the Classroom, rather than as an isolated system. Within the Classroom, the speaker's possible tasks are the complete set of tasks that the Classroom can recognize. Based on the particular task the speaker is performing at the moment, Gargoyle will have different vision tasks that it needs to focus on and have different techniques it should use to accomplish them. For instance, as the speaker is walking around and lecturing, Gargoyle needs to accurately track the location of the speaker's head, so that the Classroom's presentation camera stays trained on his head. The locations of the speaker's hands are less important, so Gargoyle can devote less effort to tracking them. Furthermore, when Gargoyle is using a background-subtraction technique to track the speaker's position, it continually updates its representation of the background to compensate for minor lighting changes and sensory noise. But, when the speaker is standing still, Gargoyle stops updating its background representation (otherwise the speaker will slowly bleed into the background), producing a more accurate segmentation for the speaker's extremities.

The key point here is that the visual context allows Gargoyle to tune its techniques to be more efficient (based on what is important for the current task) and to be more reliable (by taking advantage of the constraints provided by the current task context).

In all these examples, the key idea is that if the designers of computer systems reason deeply about how their users will use their systems, they will be able to implement systems that take advantage of the various constraints inherent in those tasks. Such a system is able to reason about their user's actions within the context of how it expects to be used. For example, the Intelligent Classroom has a fixed set of speaker activities that it reasons about and, when the speaker does something, the Classroom expects that he is involved in one of the activities it knows about. The Classroom's representation of these activities serves as the task knowledge that it uses to understand the speaker actions. The Classroom does not have to be smart about everything; it only needs to understand the ways it is going to be used.

7.3 More open issues

At the end of some of the chapters in this dissertation, I discussed some open issues suggesting interesting future work relevant to the topics of those chapters. In this section, I look at some other issues that do not fit into any chapter in particular. The first looks at how the Intelligent Classroom could be made more effective through personalization and the second addresses the paradox of privacy versus helpfulness.

7.3.1 Meeting the needs of different users

There is probably no universally perfect set of Intelligent Classroom behaviors. What would be an ideal interaction with the Classroom for one speaker would likely be

impossibly frustrating for another. While one speaker might appreciate the Classroom proactively aiding him, another might wish to directly control it. While one speaker may wish the Classroom to automatically advance slides when he pauses, another may not. There are two major difficulties that have to be addressed in order to personalize the Classroom for particular speakers.

First, the plan and process representations must be made flexible in such a way that they incorporate the preferences of all the speakers, but, when run, utilize only the preferences of the current speaker. Ideally, the plan and process representations would be parameterized so that the same plans and processes would be used with all speakers, but based on the identity of the speaker, would be used differently. In processes, this would involve controlling how step actions were performed (e.g. changing numeric constants) and even determining which steps were performed and in what order (e.g. disabling particular wait-for clauses). These sorts of changes could be implemented using the memory system to store the preferences as database entries–the process would query the preferences and the process actions and wait-for clauses would access the resulting variable bindings. In plans, the personalization would involve determining which speaker processes would be considered as possible explanations for speaker actions. Once again, these preferences would probably be most easily dealt with using the memory system. In this case, a process' context clause would access the speaker's preferences to determine whether it should consider the process as a potential explanation. Hopefully, these techniques could go a long way towards achieving a good degree of personalization–they do not require any changes to the Classroom's representations. However, they could not easily deal with a situation where the speaker wishes to change what plan is used to cooperate with

a particular speaker plan. Also, as personalization information is added to the plan and process representations, they get harder to understand. There is a great deal of room for future work in this area.

Second, the Classroom must learn the speaker's preferences. This involves at least three ways that the speaker might demonstrate his preferences. The speaker might, in the course of giving several presentations, demonstrate to the Classroom which plans he does or does not use, allowing the Classroom to strongly favor his most commonly used plans. The speaker might, during a presentation, use commands to inform the Classroom when it does something contrary to his wishes. Finally, the speaker might, through a separate user interface tool, dictate his preferences to the Classroom. All three of these techniques would require that significant additional functionality be added to the Classroom.

7.3.2 Privacy (or Need we fear Big Brother?)

In thinking about computer systems that observe people, trying to understand what they are doing, there is a tension between enjoying the benefits of having systems be helpful because they understand enough about what their users do, and fearing that such systems may be used to invade their privacy. After all, there seems to be a fine line separating trying to understand what someone is doing and just spying on them. With the help of an active imagination, it is not hard to mentally transform the cooperative approaches discussed in this dissertation into the Orwellian telescreen used to menace the people of 1984 [63] into submission. Should we worry that our cooperative agents will become a sort of Big Brother, monitoring our every action and ratting us out for our "thought crimes"?

I do not see there being any potential for this to become a problem any time soon.

The level of understanding that cooperative agents such as the Classroom maintain is both too high and too low for the sort of surveillance that we should be worried about. Interactive agents abstract away the low-level details needed to understand precisely what the user is doing (e.g. trying to find the precise ratio of nitric acid to sulfuric acid needed in producing nitroglycerin), but also ignore the ultimate reasons behind what their users are doing (e.g. preparing to commit a terrorist act). The reason is that these two levels of understanding are far too rich. In order to understand the low-level details, an agent must know about everything a user might do, at an extremely detailed level. This would require a massive knowledge engineering effort! Similarly, in order to understand the possible motives behind a user's actions, an agent must also be able to infer the precise motives behind a set of user actions. While at the middle level abstraction there are a manageable number of possible explanations, at either extreme they are countless.

Any would-be Big Brother needs to be able to reason at both extremes to determine a person's ultimate intentions and gather enough details to support this explanation. Otherwise, such a surveillance system is no more effective than a system like Echelon [14, 59], believed to be used by the NSA to scan e-mail and other communications, looking for particular words and phrases to determine whether a particular message warrants using a human inspector to determine if the author has malicious intentions. A software system designer has little incentive to build the sort of cooperative system that could be used in this way because such efforts do little to improve the usefulness of a system. While a user may find the Assistant in Microsoft Word quite helpful when it recognizes that he is writing a letter and offers to automatically format it for him, he would probably not expect it to say, "It looks

like you are planning to blow up such-and-such a building. Would you like help?" A given individual high-level plan is likely to happen very rarely, so a great deal of engineering effort produces an imperceptible increase in usability.

Instead of wasting a great deal of effort to add basically unwanted features, system designers are led to devote time to allowing their systems to reason about the sorts of tasks they are intended to be used for. Most of this reasoning will take place in the middle levels of abstraction where a moderate amount of knowledge engineering produces a good deal of improvement. Efforts towards building a good snoop do not produce a more competent assistant and, as such, are unlikely to be undertaken by the designers and implementers of cooperative agents.

7.4 Codetta

In music, the coda is short passage of music tacked on to the end of a musical work, making a more effective ending. Often the coda serves as a dramatic summing up of the whole piece: an exciting climax to remember. A codetta is a brief, and perhaps less ambitious, coda–a response to the musical question, "Now how do I end this thing?"

So: this dissertation describes a competent assistant that advances the state of the art for physical interaction: the Intelligent Classroom, a prototype automated lecture facility that serves as its own audio/visual assistant. The Classroom domain provides an interesting level of interaction and also exhibits many of the interesting challenges of mobile autonomous robotics research. Key contributions of this research include:

- Techniques for connecting the low-level sensors and actuators to a plan-recognizing

symbolic execution system. The Intelligent Classroom's plan and process representations are grounded in events and quantities that can be directly sensed by its sensors. The Classroom's context-dependent camera-work demonstrates the application of plan recognition results to the physical control of a pan-tilt-zoom camera.

- Demonstrating how a good level of intentional cooperative interaction can be achieved without explicitly reasoning about goals. Each of the Classroom's plans represents a common understanding of how a speaker and the Classroom should interact when the speaker pursues a particular course of action. A plan includes the actions the speaker can be expected to take, the actions the Classroom should take to cooperate, and a set of synchronization constraints to ensure that the actions occur at the right times.

- A voice-operated slide-advancing system. Jabberwocky uses speech-recognition results to determine what slide a speaker wishes to discuss, allowing a speaker to dynamically change his presentation to suit the needs of his audience. This system uses a probabilistic approach to deal with noisy speech recognition results that are often less than 50% accurate.

Cadence.

Bibliography

[1] Gregory D. Abowd, Chris Atkeson, Ami Feinstein, Cindy Hmelo, Rob Kooper, Sue Long, Nitin "Nick" Sawhney, and Mikiya Tani, *Teaching and learning as multimedia authoring: The Classroom 2000 project*, ACM Multimedia '96 Conference, 1996.

[2] James Allen, *Planning as temporal reasoning*, Second International Conference on Principles of Knowledge Representation and Reasoning, 1991.

[3] James Allen, Bradford Miller, Eric Ringger, and Teresa Sikorski, *Robust understanding in a dialogue system*, 34th Meeting of the Association for Computational Linguistics, 1996.

[4] José A. Ambros-Ingerson and Sam Steel, *Integrating planning, execution and monitoring*, 15th National Conference on Artificial Intelligence, 1998.

[5] American Association for Artificial Intelligence, *AAAI 1998 spring symposium on intelligent environments. AAAI TR SS-98-02*, 1998.

[6] Isaac Asimov, *I, Robot*, Doubleday, 1950.

[7] Tucker Balch, *The AAAI mobile robot challenge*, http://www.coral.cs.cmu.edu/-aaairobot/challenge, 1999.

[8] Aaron Bobick, Stephen Intille, Jim Davis, Freedom Baird, Claudio Pinhanez, Lee Campbell, Yuri Ivanov, Arjan Schutte, and Andrew Wilson, *The Kidsroom: A perceptually-based interactive and immersive story environment*, Presence: Teleoperators and Virtual Environments **8** (1999), no. 4, 367–391.

[9] _____, *Design decisions for interactive environments: Evaluating the Kidsroom*, AAAI 1998 Spring Symposium on Intelligent Environments. AAAI TR SS-98-02, 1998.

[10] Shannon Bradshaw, Andrei Scheinkman, and Kristian Hammond, *Guiding people to information: Providing an interface to a digital library using reference as a*

basis for indexing, Fourth International Conference on Intelligent User Interfaces, 2000.

[11] Jason A. Brotherton and Gregory D. Abowd, *Rooms take note: Room takes notes*, AAAI 1998 Spring Symposium on Intelligent Environments. AAAI TR SS-98-02, 1998.

[12] Jay Budzik and Kristian J. Hammond, *User interactions with everyday applications as context for just-in-time information access*, Fourth International Conference on Intelligent User Interfaces, 2000.

[13] Wolfram Burgard, Armin B. Cremers, Dieter Fox, Dirk Hahnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun, *The interactive museum tour-guide robot*, 15th National Conference on Artificial Intelligence, 1998.

[14] Duncan Campbell, *Careful, they might hear you*, The Age (http://-www.theage.com.au/daily/990523/news/news3.html) (1999).

[15] Karel Capek, *R.U.R. (Rossum's universal robots): a fantastic melodrama*, Doubleday (English translation), 1920.

[16] Lewis Carroll, *Through the looking glass*, 1871.

[17] David Chapman, *Planning for conjunctive goals*, Artificial Intelligence **32** (1987), 333–377.

[18] Eugene Charniak and Robert P. Goldman, *A Bayesian model of plan recognition*, Artificial Intelligence **64** (1993), 53–79.

[19] Michael H. Coen, *Building brains for rooms: Designing distributed software agents*, Ninth Conference on Innovative Applications of Artificial Intelligence, 1997.

[20] _____, *Design principles for intelligent environments*, 15th National Conference on Artificial Intelligence, 1998.

[21] Michael H. Coen and Kevin W. Wilson, *Learning spatial event models from multiple-camera perspectives in an intelligent room*, 25th Annual Conference of the IEEE Industrial Electronics Society, 1999.

[22] Andrew Crossen, Jay Budzik, Mason Warner, Larry Birnbaum, and Kristian J. Hammond, *XLibris: An automated library research assistant*, Fifth International Conference on Intelligent User Interfaces, 2001.

[23] Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld, *A scalable comparison-shopping agent for the world-wide web*, First International Conference on Autonomous Agents, 1997.

[24] Brian Drabble, *EXCALIBUR: a program for planning and reasoning with processes*, Artificial Intelligence **62** (1993), 1–40.

[25] Charles Earl and R. James Firby, *Combined execution and monitoring for control of autonomous agents*, First International Conference on Autonomous Agents, 1997.

[26] Stephanie Everett, Kenneth Wauchope, and Manuel Perez-Quinones, *Creating natural language interfaces to VR systems: Experiences, observations and lessons learned*, Fourth International Conference on Virtual Systems and Multimedia, 1998.

[27] George Ferguson and James F. Allen, *TRIPS: An integrated intelligent problem-solving assistant*, 15th National Conference on Artificial Intelligence, 1998.

[28] R. E. Fikes and N.J. Nilsson, *STRIPS: A new approach to the application of theorem proving to problem solving*, Artificial Intelligence **2** (1971), 189–208.

[29] R. James Firby, *Adaptive execution in complex dynamic worlds*, Ph.D. thesis, Yale University, January 1989.

[30] ———, *The RAP language manual*, Tech. report, Animate Agent Procect Working Note AAP-6, University of Chicago, 1995.

[31] R. James Firby, Roger E. Kahn, Peter N. Prokopowicz, and Michael J. Swain, *An architecture for vision and action*, 14th International Joint Conference on Artificial Intelligence, 1995.

[32] R. James Firby, Peter N. Prokopowicz, Michael J. Swain, Roger E. Kahn, and David Franklin, *Programming CHIP for the IJCAI-95 robot competition*, Artificial Intelligence Magazine **17** (1995), 71–81.

[33] Will Fitzgerald, *Building embedded conceptual parsers*, Ph.D. thesis, Northwestern University, March 1995.

[34] Joshua Flachsbart, *Gargoyle: Vision in the Intelligent Classroom*, Master's thesis, University of Chicago, 1997.

[35] Joshua Flachsbart, David Franklin, and Kristian J. Hammond, *Improving human computer interaction in a classroom environment using computer vision*, Fourth International Conference on Intelligent User Interfaces, 2000.

[36] David Franklin, Shannon Bradshaw, and Kristian J. Hammond, *Jabberwocky: You don't have to be a rocket scientist to change slides for a hydrogen combustion lecture*, Fourth International Conference on Intelligent User Interfaces, 2000.

[37] David Franklin, Michael J. Swain, Roger E. Kahn, and R. James Firby, *Happy patrons make better tippers: Creating a robot waiter using Perseus and the animate agent architecture*, International Conference on Automatic Face and Gesture Recognition, 1996.

[38] Erann Gat, *Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots*, Tenth National Conference on Artificial Intelligence, 1992.

[39] Michael P. Georgeff and Amy L. Lansky, *Reactive reasoning and planning*, Sixth National Conference on Artificial Intelligence, 1987.

[40] Abigail Gertner, Bonnie Webber, and John R. Clarke, *Recognizing and evaluating plans with diagnostic actions*, IJCAI 1995 Workshop on The Next Generation of Plan Recognition Systems, 1995.

[41] David Gustafson, *AAAI 2001 hors d'oeuvres contest*, http://www.cis.ksu.edu/-˜dag/aaai2001/hdcontest.html, 2001.

[42] David Heckerman and Eric Horvitz, *Inferring informational goals from free-text queries: A Bayesian approach*, 14th Conference on Uncertainty in Artificial Intelligence, 1998.

[43] Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse, *The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users*, 14th Conference on Uncertainty in Artificial Intelligence, 1998.

[44] Marcus J. Huber, Edmund H. Durfee, and Michael P. Wellman, *The automated mapping of plans for plan recognition*, Tenth Conference on Uncertainty in Artificial Intelligence, 1994.

[45] Marcus J. Huber and Tedd Hadley, *Multiple roles, multiple teams, dynamic environment: Autonomous Netrek agents*, First International Conference on Autonomous Agents, 1997.

[46] IBM, *Viavoice: Voice systems*, http://www.ibm.com/software/speech, 2001.

[47] Roger E. Kahn, *Perseus: An extensible vision system for human-machine interaction*, Ph.D. thesis, University of Chicago, 1996.

[48] Roger E. Kahn and Michael J. Swain, *Understanding people pointing: The Perseus system*, International Symposium on Computer Vision, 1995.

[49] Roger E. Kahn, Michael J. Swain, Peter N. Prokopowicz, and R. James Firby, *Gesture recognition using the Perseus architecture*, Computer Vision and Pattern Recognition, 1996.

[50] Gal A. Kaminka and Milind Tambe, *What is wrong with us? Improving robustness through social diagnosis*, 15th National Conference on Artificial Intelligence, 1998.

[51] Henry A. Kautz and James F. Allen, *Generalized plan recognition*, National Conference on Artificial Intelligence, 1986.

[52] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, and Itsuki Noda nad Eiichi Osawa, *RoboCup: The robot world cup initiative*, IJCAI 1995 Workshop on Entertainment and AI/Alife, 1995.

[53] Kurt Konolige and Martha E. Pollack, *Ascribing plans to agents: Preliminary report*, 11th International Joint Conference on Artificial Intelligence, 1989.

[54] David Kortenkamp, Eric Huber, and R. Peter Bonasso, *Recognizing and interpreting gestures on a mobile robot*, 13th National Conference on Artificial Intelligence, 1996.

[55] Nicholas Kushmerick, Steve Hanks, and Daniel Weld, *An algorithm for probabilistic least-commitment planning*, 12th National Conference on Artificial Intelligence, 1994.

[56] John E. Laird, Alan Newell, and Paul S. Rosenbloom, *Soar: An architecture for general intelligence*, Artificial Intelligence **33** (1987), 1–64.

[57] Neal Lesh and Oren Etzioni, *Insights from machine learning for plan recognition*, IJCAI 1995 Workshop on The Next Generation of Plan Recognition Systems, 1995.

[58] Henry Lieberman, *Letizia: An agent that assists web browsing*, 14th International Joint Conference on Artificial Intelligence, 1995.

[59] Greg Lindsay, *The government is reading your e-mail: With the Echelon program, the U.S. and its allies are collaborating to monitor the Internet*, Time Digital (http://www.time.com/time/digital/daily/0,2822,27293,00.html) (1999).

[60] Mark Lucente, Gert-Jan Zwart, and Andrew D. George, *Visualization space: A testbed for deviceless multimodal user interface*, AAAI 1998 Spring Symposium on Intelligent Environments. AAAI TR SS-98-02, 1998.

[61] Pattie Maes, *Agents that reduce work and information overload*, Communications of the ACM **37** (1994), no. 7, 31–40.

[62] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller, *Introduction to WordNet: An on-line lexical database*, Journal of Lexicography **3** (1990), no. 4, 234–244.

[63] George Orwell, *1984*, Harcourt Brace Jovanovich, Inc., 1949.

[64] J. Scott Penberthy and Daniel Weld, *UCPOP: A sound, complete, partial-order planner for ADL*, Third International Conference on Principles of Knowledge Representation and Reasoning, 1992.

[65] Claudio Pinhanez and Aaron F. Bobick, *Intelligent studios: Using computer vision to control TV cameras*, IJCAI 1995 Workshop on Entertainment and AI/Alife, 1995.

[66] Claudio S. Pinhanez and Aaron F. Bobick, *Approximate world models: Incorporating qualitative and linguistic information into vision systems*, 13th National Conference on Artificial Intelligence, 1996.

[67] Peter N. Prokopowicz, Roger E. Kahn, R. James Firby, and Michael J. Swain, *Gargoyle: Context-sensitive action vision for mobile robots*, 13th National Conference on Artificial Intelligence, 1996.

[68] Bradley J. Rhodes, *Margin notes: Building a contextually aware associative memory*, Fourth International Conference on Intelligent User Interfaces, 2000.

[69] Charles Rich and Candace Sidner, *COLLAGEN: When agents collaborate with people*, First International Conference on Autonomous Agents, 1997.

[70] Charles Rich, Candace L. Sidner, and Neal Lesh, *COLLAGEN: Applying collaborative discourse theory to human-computer interaction*, Artificial Intelligence Magazine **22** (2001).

[71] Earl D. Sacerdoti, *The nonlinear nature of plans*, Fourth International Joint Conference on Artificial Intelligence, 1975.

[72] Roger Schank and Robert Abelson, *Scripts plans goals and understanding*, Lawrence Erlbaum Associates, Hillside, NJ, 1977.

[73] M. J. Schoppers, *Universal plans for reactive robots in unpredictable environments*, Tenth International Joint Conference on Artificial Intelligence, 1987.

[74] Milind Tambe, *Agent architectures for flexible, practical teamwork*, 14th National Conference on Artificial Intelligence, 1997.

[75] Milind Tambe and Paul S. Rosenbloom, *RESC: An approach for real-time, dynamic agent tracking*, 14th International Joint Conference on Artificial Intelligence, 1995.

[76] Bonnie Webber, Ron Rymon, and John R. Clarke, *Flexible support for trauma management through goal-directed reasoning and planning*, Artificial Intelligence in Medicine **4** (1992), 145–163.

[77] Robert Wilensky, *Inside computer understanding*, ch. 7, Lawrence Erlbaum Associates, Inc., 1981.

[78] David Wilson and Shannon Bradshaw, *CBR textuality*, Fourth UK Case-Based Reasoning Workshop, 1999.

[79] William A. Woods, *Understanding subsumption and taxonomy: A framework for progress*, Principles of Semantic Networks: Explorations in the Representation of Knowledge, Morgan Kaufmann, 1991.