Northwestern University

The Institute for the Learning Sciences

LEARNING FROM STORIES: INDEXING AND REMINDING IN A SOCRATIC CASE-BASED TEACHING SYSTEM FOR ELEMENTARY SCHOOL BIOLOGY

Technical Report # 43 • July 1993

Daniel Choy Edelson

Established in 1989 with the support of Andersen Consulting

NORTHWESTERN UNIVERSITY

Learning from Stories: Indexing and Reminding in a Socratic Case-Based Teaching System for Elementary School Biology

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By
Daniel Choy Edelson

EVANSTON, ILLINOIS

June 1993

This work was supported in part by a grant from the IBM Program for Innovative Uses of Information Technology in K-12 education and by the Defense Advanced Research Projects Agency monitored by the Office of Naval Research under contract N00014-91-J4092. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting. It receives additional support from Ameritech and North West Water, Institute partners.

Abstract

Learning from Stories: Indexing and Reminding in a Socratic Case-Based Teaching System for Elementary School Biology

Daniel Choy Edelson

Good teachers teach with stories. To reproduce this effective teaching technique in a computer-based learning environment, we have developed the *case-based teaching architecture.* A case-based teaching system uses artificial intelligence techniques adapted from case-based reasoning research to teach by presenting stories in context. That context is established by a *task environment* which provides a student with a naturally motivating task and an engaging environment for pursuing that task. A *storyteller* monitors the student's interactions with the task environment, looking for opportunities to present stories that will help the student to learn from his or her situation. Learning from stories provides students with cases that support the natural process of case-based reasoning.

The challenge of constructing a case-based teaching system lies in developing a scheme for indexing the stories in the storyteller's library and in implementing algorithms that will allow the storyteller to identify appropriate stories when they are relevant. To retrieve these stories, a case-based teaching system employs *reminding strategies.* Reminding strategies enable a storyteller to identify and retrieve stories to serve specific goals. Reminding strategies rely on a sufficiently expressive indexing scheme for labeling the stories in a story library.

In this dissertation, I present the case-based teaching architecture and propose several general reminding strategies. These have been used in the development of Creanimate, a case-based teaching system that teaches animal adaptation to elementary school students . Its task environment invites a student to create a new animal and engages the student in a discussion of how his or her invented animal might survive. Its storyteller possesses a library of dramatic video clips showing actual animals in the wild. These video stories are presented as illustrations of the principles that arise in the discussion of the student's animal. Creanimate uses thought-provoking questions called *explanation questions* as the basis for dialogues that establish a context for learning from stories. Creanimate employs three general reminding strategies: example reminding, similarity-based reminding, and expectation-violation reminding. Its story library is indexed according to the principles of animal adaptation that the stories illustrate.

Acknowledgements

I've always considered the acknowledgements to be the most interesting part of any dissertation. It is the one part of the dissertation where you get a feeling for the writer as a person with a life, family, and friends, and an academic environment as being a community of individuals with personalities. So, I hope you will indulge me if I go on longer than I probably should.

I wrote the first draft for this acknowledgement in October 1988 in my second month in graduate school. I have that draft in front of me now as I write this. At that time, I did not know what my research was going to be or even who my advisor would be, but I had already started to think about the support that had been necessary to get me to that point. That acknowledgement went on at great length about the excellent teachers I have had at every level and how they shaped me and my research interests. I had the good fortune to attend one of the best public school systems in the country, and my experience left me a strong believer in the potential of public schools. I realize my experience is the exception not the rule, but it is an important part of the reason I am committed to a career of research in improving education.

The Institute for the Learning Sciences has been a terrific environment in which to be a grad student. It is an exciting place, full of bright people who care deeply about their work. The credit for creating this environment goes entirely to Roger Schank. An effective organization requires a strong vision, which ILS gets from Roger's bold theories, philosophies, and personality. They provide ILS with its life and its liveliness.

Roger's role in my time here has gone far beyond simply establishing an environment for me to work in. He is a wonderful teacher who knows how to set a balance between giving a student space to explore and a structure to hang on to. He takes the advising part of the advisor role extremely seriously. In fact, advising is the point where his academic interest in stories and his natural gift for storytelling come together. He gives great advice, and it invariably comes in the form of a story. I intend to continue to rely on him for advice for as long as he'll tolerate me.

Other faculty members have also played significant roles in my academic and personal development. It appears that no dissertation written in our lab in the last decade fails to mention Larry Birnbaum as a key influence in the writer's experience. I am no exception. I owe my interest in AI to Larry who gave me my first taste for it while I was an undergraduate at Yale. He has an enthusiasm for the issues of AI that is irresistible and infectious. In the same way that Larry gave me my grounding in the issues of AI, Allan Collins introduced me to the issues of education. He has been a valuable supporter of mine over the past three years and has been there when I needed him for insight and guidance. Bill Purves, professor of biology and consultant on matters biological, deserves much credit for shaping this research early in its history and for helping me keep perspective through his sociological insights from afar. Two faculty members that served as lab heads for the Creanimate project also gave me valuable input and support. They are Chris Riesbeck and Ray Bareiss. Alex Kass, though he never played any official role in my education or research has been a constant source of ideas and advice.

Faculty members play an important role in the life of a grad student, but it's the other grad students who are down in the trenches with you who carry you through. I've had a string of great officemates who were all able to guide me through the strange culture which I have joined. Chris Owens, Bill Ferguson, the Erics Jones and Domeshek, and Dick Osgood are all older and wiser than I, and helped me through various mine fields.

I believe the members of my entering class share a particularly strong bond because, in addition to the usual trials of classes and quals that we endured together, we were the adventurous first-years who pulled up our newly planted roots from New Haven and made the trek to Evanston. Ours are the first Ph.D.'s to be granted at ILS, and we have had the opportunity to set precedents, rather than follow those set by others. I think Matt Brand, Michael Freed, Kemi Jona, Enio Ohmaye, Dick Osgood, Louise Pryor, and I will always think of ourselves as the trailblazers.

I have made friends and established important professional relationships with dozens of grad students and staff members at ILS, as well as their spouses. Some of the people I feel most indebted to are Andy Fano, Rebecca Handel-Fano, Diane Schwartz, Liz Gearen, Laura Reichert, Tracey McCurrach, Doug MacFarlane, Judith Fraivillig, and Ben Bell. In addition, I would like to thank the non-academic members of the ILS community who have played central roles in getting me through my years here: Michelle Suran, Elizabeth Brown, Teri Lehmann, Celia McNaughton, Grant Beadle and Heidi Levin. In the last six months, I have split my time between finishing this thesis and managing a new project. I appreciate the assistance and understanding of Roy Pea and the rest of the CoVis team during this time.

My family also deserves acknowledgement. I've been very fortunate in my life, but with the few obstacles I have had to face, I've had tremendous, loving support from my them. My parents laid the groundwork, but it is my wife Vivian who has been there with me through each and every day of getting a Ph.D. Vivian has been a constant source of strength and has made even the hardest stages of this process a joy. In the years that I've been in graduate school, my family has doubled in size. In addition to the brothers that I started out with, I now have sisters-in-law and parents-in-law. My brothers, their wives, and my in-laws have all helped me to get through in ways big and small.

My final (non-financial) acknowledgement goes to the other people who worked on Creanimate. One of the great things about ILS is the opportunity, as a grad student, to work with a team of skilled and dedicated professionals. They are the ones who did the real work described in this research. I mainly got it started and tried to keep it all together. The lead programmer on Creanimate over the years has been Riad Mohammed, who proved able to do anything I could do, only better. Diane Schwartz was responsible for the interface design and the artwork. Other programmers who worked on the user interface at various stages include Charles Earl, David Newton, and Ken Greenlee. Bob Kaeding was the indexer who put the knowledge in Creanimate. He was the final link that enabled us to put the whole thing together. Several other grad students worked on Creanimate in various capacities. Ben Bell wrote Creanimate's first parser. Will Fitzgerald wrote its current parser. Ian Underwood gathered and partially indexed the first set of video clips. John Cleave wrote the code for one of the most important dialogue plans.

Finally, research does not happen without funding. In my case, I received personal support from Defense Advanced Research Projects Agency monitored by the Office of Naval Research under contracts N00014-91-J4092 and N00014-90-J-4117. The Creanimate project itself was funded by a grant from IBM through their Program for Innovative Uses of Information Technology in K-12 Education. I think their willingness to fund experimental educational systems shows an authentic concern for the future of education in our country beyond their own business interests. I am also very grateful to the National Geographic Society and the Encyclopaedia Britannica Educational Corporation for allowing us to use their footage in Creanimate. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting. It receives additional support from Ameritech and North West Water, Institute Partners.

Table of Contents

List of Illustrations

List of Tables

[This page intentionally left blank]

Chapter 1
Introduction

Suppose you could create a new animal by taking an existing animal and changing it some way. What would you make?

This invitation is offered to children by Creanimate, the computer-based learning environment which is the subject of this dissertation. The student's response to this question forms the starting point for a discussion between child and computer in which they collaborate on a design for a new animal. Their dialogue is accompanied by dramatic video clips of real animals in the wild that illustrate the principles of animal adaptation. To set the stage for the discussion of Creanimate, I would like you to consider the following question.

Suppose you could create a new computer-based learning environment by taking emerging technologies and applying them in new ways. What would you make?

If you were an educator, you might say:

It should be responsive.
It should help children to learn to ask questions.
It should get children interested in new subjects.
It should help children to learn from their mistakes.
It should challenge children to think for themselves.
It should let children explore at their own pace.
It should encourage creativity.

If you were a researcher in computer science, you might say:

It should have data structures for encoding the subject matter.
It should have algorithms that implement effective teaching strategies.
It should have a clear, easy to use interface.
It should take advantage of a computer's graphical and multimedia capabilities.

If you were a parent you might say:

It should help my child develop self-confidence.
It should encourage my child to be him or herself.
It should help my child to develop skills that will enable him or her to be successful in life.

If you were a researcher in education you might say:

It should operate according to an underlying theory of instruction.
It should have empirically demonstrable results.

If you were a child you might say:

It should be fun.
It should be easy to use.
It should let me be in charge.
It should have flashy graphics and video.
It should teach me things I am really interested in.
It should be fun.

In developing the Creanimate system, we have taken into account the perspectives of all the constituencies above: educators, parents, computer scientists, educational researchers, and children. The task we have set for ourselves is to create a learning environment that satisfies all their requirements, especially those of the child. For if children don't like it, a learning environment should not be considered successful.

1.1 Goals

The requirements for effective teaching can be summed as three criteria: correspondence, engagement, and responsiveness. These criteria express the fundamental goals of this research.

Correspondence. Teaching methods must correspond to the natural learning mechanisms of the student.
Engagement. In the course of learning, a student must be involved in the pursuit of personally meaningful goals. Engagement provides a student with a framework for integrating knowledge and a motivation to learn. The direct result of engagement is empowerment.
Responsiveness. Instruction must respond immediately to the actions, interests, and concerns of each student.

Admittedly, this is an ambitious task. Most researchers concern themselves with only one of the constituencies listed above, and many do not achieve even one of these criteria for effective teaching. However, the future of education is a serious concern, and it deserves ambitious efforts.

Before we go any further, there is an important question that must be answered. The goals of correspondence, engagement, and responsiveness may be important, but why would we use the computer to reach them? The answer is that computer technologies, specifically artificial intelligence and multimedia, offer a unique opportunity to improve on current educational practice. Current practice has fallen short of these goals either because of the limitations of technology and resources or because of misplaced priorities.

Correspondence. Achieving correspondence requires that educators have a model of the way a student learns and that material be presented in a manner that is consistent with the model. Many current practices came from theories of learning based on outmoded psychological ideas; other, better theories have not made it into classroom practice as much as they deserve. Teachers and curriculum designers facing economic and political pressure often lack the resources and expertise to develop theoretically sound educational materials.

Engagement. To achieve engagement, each student must be allowed to pursue personally meaningful goals and exposed to subject matter in the pursuit of those goals. If you capture the interest of a student through goals that engage him, then he is motivated to learn the material that will help him to achieve those goals. Engagement is difficult to achieve in the context of current, highly structured curricula whose relevance to students' lives is difficult to ascertain. Too often the goals that students are presented with in order to motivate their learning are not meaningful to the student. The worst motivational goal is, of course, good grades, but even educational designers who appreciate, in principle, the importance of engagement have produced silly games in which the student must do things like solve arithmetic problems in order to slay a giant. In these contrived games, solving the problem is irrelevant to any personally meaningful goal of the student's. The goal a student pursues in learning must be both intrinsically motivating to the student and be sensible within the context of the subject matter.

Responsiveness. To achieve responsiveness, the presentation of material must be flexible enough to respond to the specific interests of any individual student. The ability to achieve responsiveness is severely limited in the traditional classroom setting. In a classroom with more than a handful of students it is impossible for a teacher to respond to the interests of an individual student when it might conflict with the interests of the class as a whole. Although teachers today have enormous resources including photographs, films, and videos available to them, they must determine in advance what materials will be available on any given day and are thus constrained by the available materials as to how responsive they can be. If they do not have the right materials at hand, they cannot effectively pursue a student's query no matter how compelling it might be.

On the other hand, carefully designed computer-based educational systems offer the promise of fulfilling these goals. First, computer-based system can be carefully crafted around a theory of learning in order to capitalize on natural learning processes. Insights from research in artificial intelligence and cognitive science have provided us with unique views of the way people reason and learn. These theories of learning emerging from AI can form the basis for system design that achieves *correspondence.* Second, by taking advantage of the computer's ability to support sophisticated interactions and simulate complex domains, computer-based systems can *engage* students. New technologies can offer students the ability to pursue meaningful goals in complex, computer-based environments. Finally, computers have an enormous ability to be *responsive.* Because interactions with computer-based systems are individualized, computers can respond to the needs and interests of each student at his or her own level and pace. In addition, with the ability to search large data bases of information rapidly, a computerized educational system, can provide a student with instant access to information that he expresses an interest in. Furthermore, using new presentation technologies, a computer can convey information through a variety of formats including text, video, computer-generated animation, and graphics. Imagine a teacher who, when asked about the Vietnam era, could immediately call up images from the news coverage of the war to stimulate a discussion of the effect of television coverage on the political system. A computer-based educational system with interactive, multimedia technology can do just that.

In this research we are committed not just to computer-based learning environments however. We are also committed to using artificial intelligence. The idea of using artificial intelligence in educational systems is not new. In fact, as early as 1970 (Carbonell 1970b) attempts were being made to build intelligent teaching systems. Yet the history of AI in education, like so much of AI, has been one of great promises and unimpressive results. If you were committed to building effective educational systems, it would seem foolhardy to argue for the use of AI in the face of so many disappointing previous attempts. However, there are reasons that the attempts to apply AI have not yet borne fruit in many effective educational systems, and these can be avoided. First, a great deal of AI in education research is motivated by the issues of AI first and by education second. The resulting systems often ignore critical pedagogical issues such as student motivation and appropriate curricula. Second, most strategies for building AI-based educational systems rely on constructing a truly intelligent teaching system, one that not only understands the material being taught but is also able to monitor a student, determine the state of his understanding, and intervene appropriately. In other words, the system must be an expert in the domain, in student psychology, and in teaching strategies. Each one of these expertises is mostly beyond the capability of current artificial intelligence technology. Combining all three definitely is. The way researchers have responded to this problem is to select a portion of this ideal system that appears tractable and to focus on building it. The idea is that eventually all of these individual pieces can be put together in a complete, intelligent teaching system. However, it is not clear from the history of AI that piecemeal solutions like this ever can or do get integrated. More important, this research is doing nothing to benefit education during the time that it is focusing on these difficult issues in AI.

In spite of these problems, there are still good reasons to build computer-based learning environments that rely on artificial intelligence. However, doing so requires an approach that avoids the pitfalls described above. The key to avoiding these pitfalls in this research is an architecture for teaching systems we call *case-based teaching*. This architecture, described in detail in Chapter 3, is inspired by the recognition that good teachers are good storytellers, and that they convey a great deal of the content that they teach in the form of stories. Taking this idea to heart, we have developed a teaching architecture for a computer that packages most of its content knowledge in prerecorded stories. Instead of being an expert in the domain, in diagnosing student states, and in appropriate teaching strategies, a case-based teaching system only needs to be an expert in one thing: recognizing the right time to tell a story to a student.

A second reason to use AI today in building effective computer-based learning environments is that theories from AI can provide useful models of the way students learn. While the ostensible aim of artificial intelligence has always been to build intelligent machines, an important result of AI research has been new theories of human learning and understanding. One such theory is the theory of case-based reasoning (Kolodner, Simpson, and Sycara-Cyranski 1985; Riesbeck and Schank 1989; Schank 1982). The reason good teachers and parents all teach with stories is that it corresponds to the natural way people learn. When people learn either from firsthand experience or from stories, they encode these experiences in cases which they retain in their memories. When they find themselves in similar situations later on, they recall these cases and decide how to act based on their conclusions about them.

The goal of this research, therefore, has been to develop a system that uses AI technology not to produce the ultimate expert in content, teaching, and learning, but to produce a system that knows enough to present stories to students that support their natural process of learning from cases. The resulting architecture is pedagogically sound in that its methods correspond to the natural learning processes of students, it engages students by capitalizing on their own meaningful goals, and it responds promptly to their needs and interests. We have implemented this architecture in the form of a system called Creanimate that teaches students in 4-7th grades about animal adaptation.

1.2 The CreANIMate System

Children are fascinated by animals, yet they overwhelmingly find school biology boring and dry. How do schools turn fascination into distaste so effectively? The answer is that when they teach science they leave out all of the elements that make science compelling, and they replace them with the memorization of disconnected facts. The Creanimate project is designed to put the enjoyment that accompanies real scientific inquiry into an effective educational experience. What makes real science exciting is that it allows you to be creative. A scientist thinks about difficult, unresolved questions, develops possible explanations, tries to make the explanations fit the data, and revises his or her explanations accordingly. At any point, a scientist may discover something entirely new. Creanimate gives elementary school children the opportunity to engage in these same activities on a level that fits their understanding and connects with their natural fascination with animals. In the course of a session with the Creanimate program, students have the opportunity to generate creative hypotheses, consider the ramifications of their hypotheses through the exploration of open-ended questions, and practice the vital skill of reasoning from cases.

Generating a creative hypothesis

Creanimate invites a student to create his or her own animal by taking an existing animal and changing it some way. For example, a student might ask for a fish with wings or a bear that can dance. In response, Creanimate engages the student in a dialogue in which he considers the ramifications of his changes. It might ask how the student's fish will use its wings or what good it might do a bear to dance. Creating a new animal is a tremendously inviting task for children because it offers them the opportunity to be whimsical and imaginative. Unlike most school activities it rewards rather than punishes their natural inclination to push beyond the limits that are constantly being imposed on them. However, as odd as it might sound, creating their own animal is a way of bringing authentic scientific practice to a level that connects with children's natures. When a scientist studies a phenomenon, one of the first things he does is perturb the system under study from its natural state and observe the effects of the perturbation. The effects of modifications on a natural system can reveal a great deal about that system in its natural state. Thus, a metallurgist studies a material by applying heat or current, bending it, stretching, and squeezing it. Creating a new animal performs the same function for a student using Creanimate. It provides him with an opportunity to learn both about the animal before it was modified and about the modification itself.

Open-ended questions

In schools, science is typically taught through the memorization of answers. Science is presented as a set of proven facts, not as the inquiry into unanswered puzzles that it really is. Following in a tradition dating back to Socrates, Creanimate helps students learn from their animals by posing thought-provoking questions. It raises open-ended questions and discusses possible answers with students. For example, suppose a student asked for a bee with a big nose. The program might respond by asking how the student's bee will use its big nose. In the ensuing discussion, the student could propose answers, e.g. to suck up honey, or ask the program to suggest answers. After discussing each possible answer, the student has the opportunity to commit to the answer for his animal. Because the answers to Creanimate's questions usually entail making additional changes to a student's animal, they frequently give rise to new questions. As a result, the student and the computer pursue an ongoing dialogue in which the student proposes changes, Creanimate raises questions, the student resolves those questions by making new changes, and the system raises new questions about the new changes. For example, suppose a student asked for a fish with wings. Creanimate would by responding why they want their fish to have wings. After considering several possibilities, the student decides his fish should use its wings to help it fly. The addition of flying raises more questions, and the student now needs to consider what else his fish will need in order to fly, as well as how flying will help his fish to survive. While the student is considering answers to its questions, Creanimate presents video clips of animals in the wild that serve as examples.

Teaching with cases

People are natural case-based reasoners. When confronted with a problem, they develop a solution by considering similar problems they've seen before. In other words, they reason from cases. In scientific inquiry, reasoning from cases is an essential skill. Therefore, Creanimate incorporates the use of cases into its dialogues with students. In doing so, it capitalizes on the powerful impact of video. The animal kingdom is overflowing with dramatic and surprising phenomena, and no medium captures this better than film. In the course of answering questions about their animals, students see video clips that show concrete examples of the answers in the form of animals in the wild. For every answer discussed, Creanimate attempts to locate a relevant video clip in its library. For example, in one session, a student who asked for a bee with a big nose answered the question "What will it use its big nose for?" with, "so it can suck up honey." Creanimate responded:

```
I think that might work.  Elephants use their big noses to
suck up liquids.  I have an interesting video about that.  If
you like baby elephants, then you'll love this video.  Would
you like to see that?"
```

In other words, Creanimate confirms appropriate with examples from its video library. On the other hand, if a student proposes an answer that Creanimate does not believe to be correct, it will respond with a video clip that shows a situation in which the answer would be correct. For example, if a student said he wanted a fish with wings to use them to dig holes, the program would respond:

```
I don't know any animals that use their wings to dig holes.
However, I can show you some things that animals use to dig
holes.
For example, meerkats use their paws to dig holes.  Would you
like to see that?
```

Creanimate helps students to resolve questions about animal adaptations by presenting them with vivid cases. These cases are used to help resolve students' questions in the same way that scientists use them to resolve the questions that arise in their work.

Creanimate in Action

The best way to describe Creanimate system is to show it. Therefore, this section contains an extended transcript showing the system in action. Creanimate has an appealing interface that allows students to express themselves through a combination of pointing and clicking with a mouse and typing. Figure 1 shows a sample screen from the program.

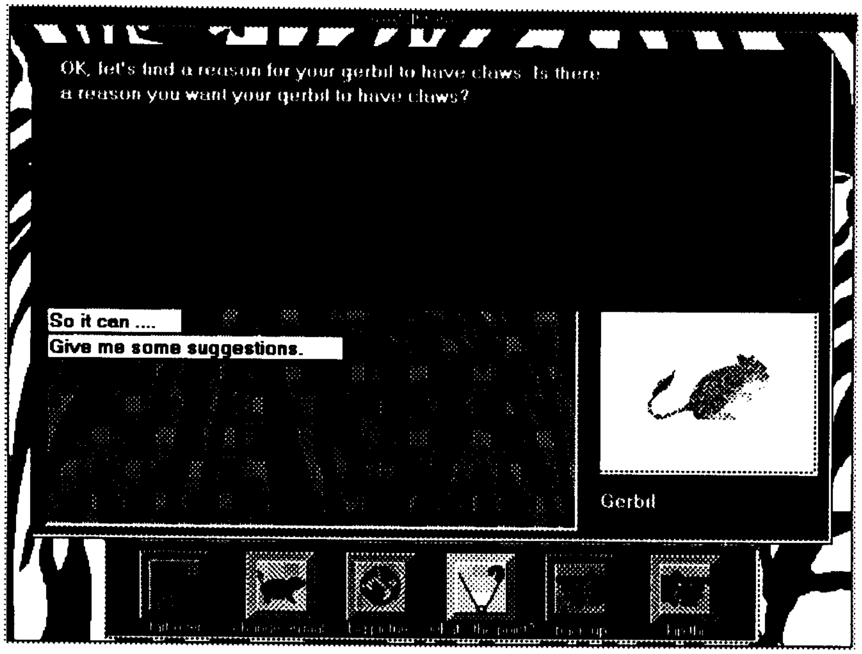

Figure 1. A sample screen from the Creanimate program

The top portion of the screen contains the program's output. The text in this section may include explanations, invitations to view video clips, or questions posed to the student. The middle section of the screen is for student input. Options are displayed in this area that students select with the mouse. If a student selects a phrase that contains an ellipsis then he or she can complete the sentence by typing in free text. At the bottom of the screen are "user control " buttons (Schank and Jona 1991), that allow the student ask for various forms of assistance (*What's the point?, Big Picture*) or exert control over the interaction (*Back up, Skip this, Change Animal, Start over*).

The following pages contain a recreation of a transcript recorded during the session of a fourth grade boy[1].

Introducing the Program

When students approach Creanimate, they see a bright title screen offering two options. They can choose to see instructions that help with the mechanics of using the program or they can start right in.

---

1 Nearly all of the examples in this dissertation are taken from transcripts of actual students using Creanimate. Transcripts from students can be identified by the presence of a transcript number. This extended example is from transcript number m2-4-10-11.56e.

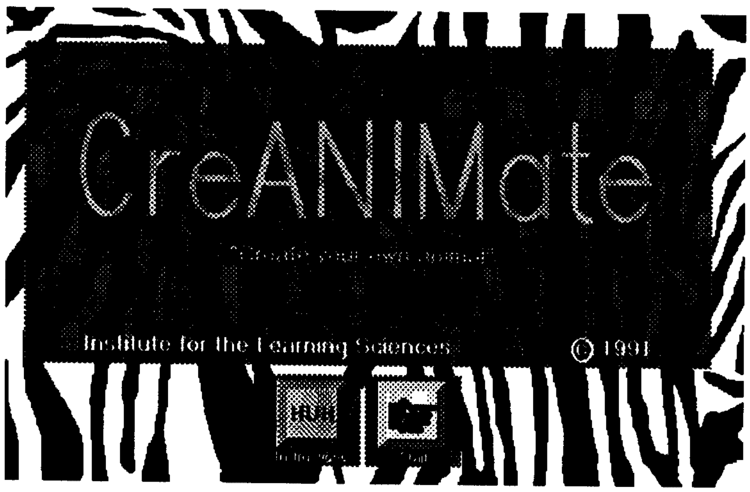

Figure 2. The title screen

The first screen students see once they start the program tells them what to expect when they use the program. The important elements are: 1) Choosing an animal; 2) Discussing the new animal; 3) Viewing videos that relate to the animal.

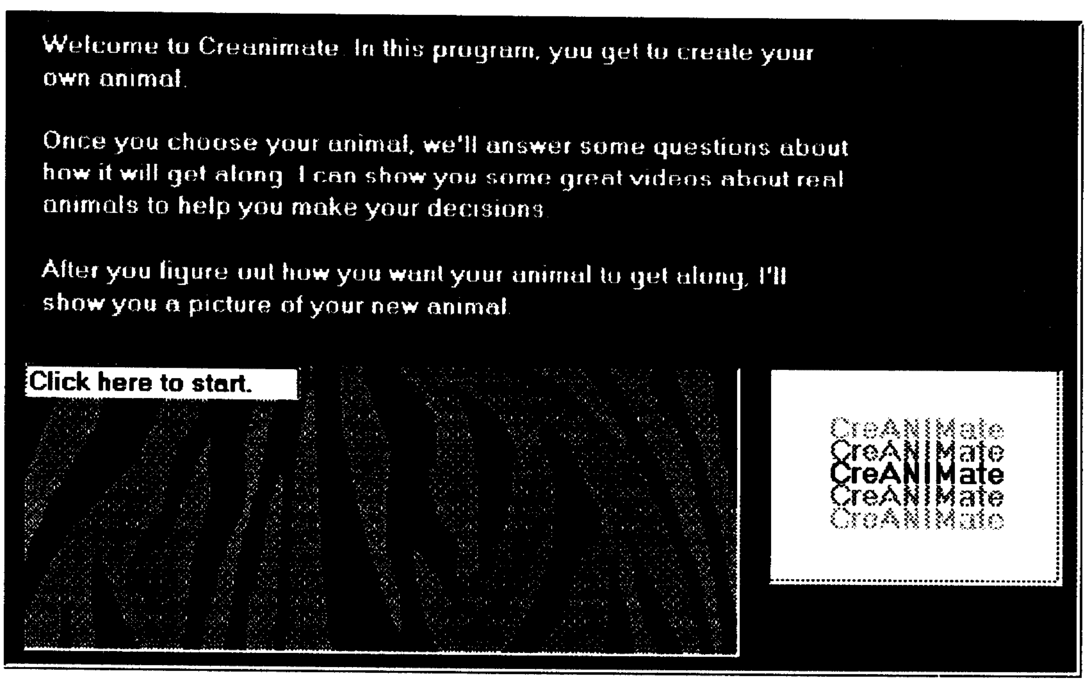

Figure 3. The introduction screen: "Welcome to Creanimate. "

Choosing an animal

The first step in Creanimate is proposing a new animal. The program presents a student with a set of options to choose from. The lists contain animals and ways to modify those animals. A student can add a physical feature to an animal or change its actions or behaviors. The animals and modifications that appear on this screen were selected based on surveys of children and availability of video.

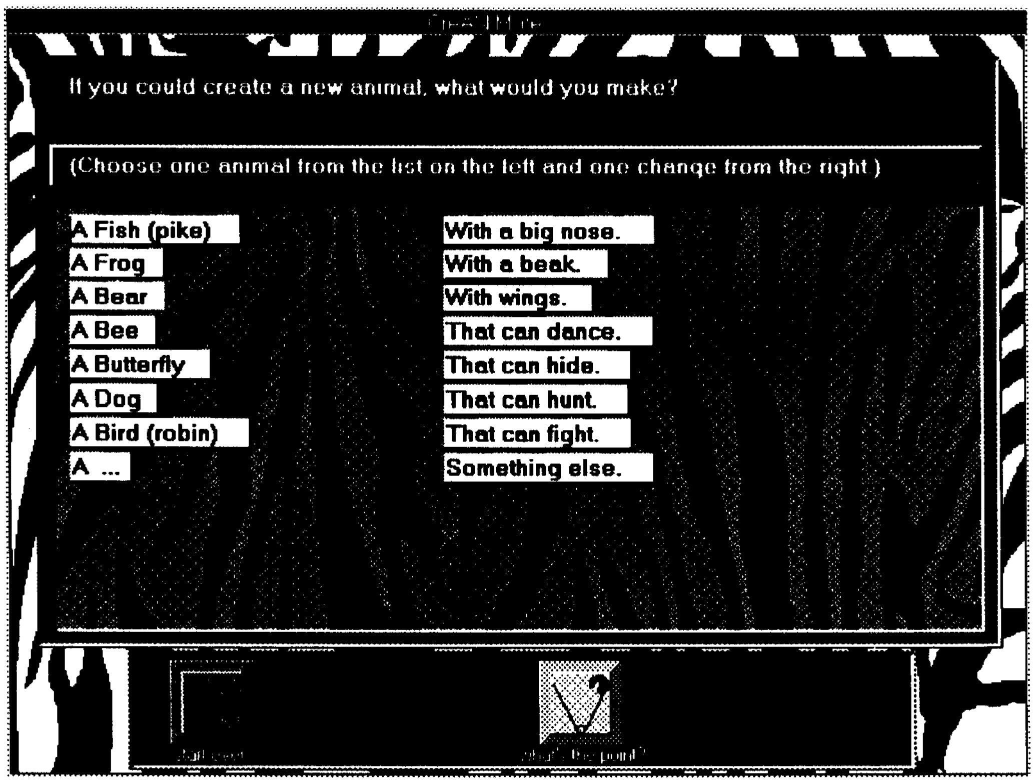

Figure 4. Choosing an animal: "If you could create a new animal, what would you make?"

By clicking first on "A Frog" and then "That can fight", the student chooses to create a frog that can fight.

STUDENT: A frog that can fight.

Ideally there would be many more choices of animals and modifications available, or there would be no choices at all, and students would just type whatever they wanted. However, the range of choices is currently limited by the size of the program's video library. As technology that supports random-access to larger libraries of video becomes available, the number of choices will be increased. Students are able to type in the name of any animal or any modification for their animal by clicking on "A ..." from the righthand column or "Something else" from the lefthand column respectively. However, since the video in the current version of Creanimate has been optimized for the choices on this screen, the discussions that result from typing in something different may not be as rich.

The Student's Animal

Before beginning the dialogue about his animal, the student gets to see a picture of it without any modifications. The animal is labeled "Before" and is accompanied by a question mark labeled "After." The "After" box represents the animal once the student has finished designing it. This screen helps establish a direction for the dialogue. The student is working toward a frog that can actually fight. When a student chooses to add one of the physical features from the first screen to his animal, he gets to see a picture of his modified animal at the end of his discussion.

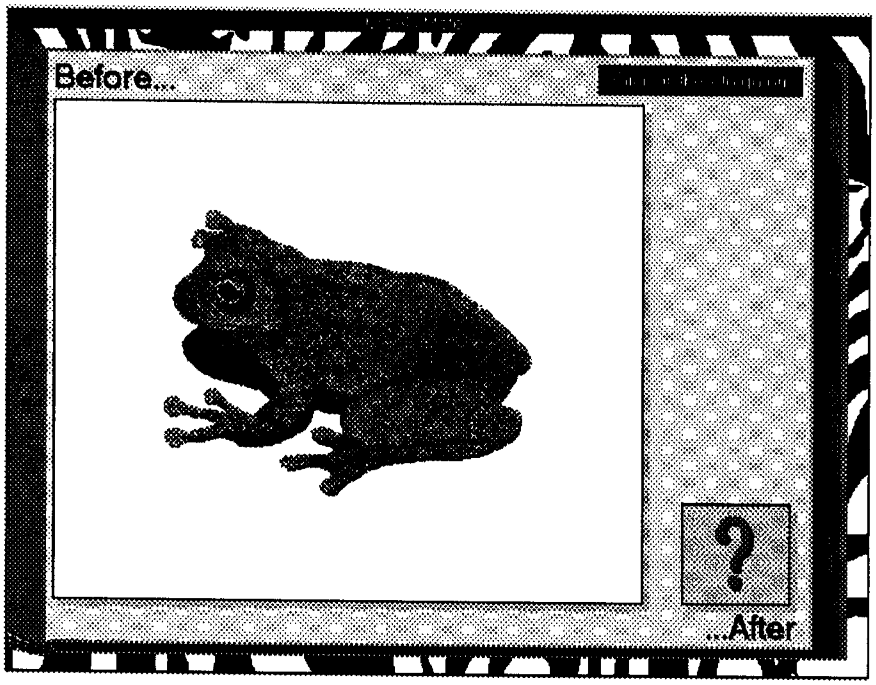

Figure 5. The "Before" picture

Posing a Question

Creanimate responds to a student's proposed animal by asking a question about it.

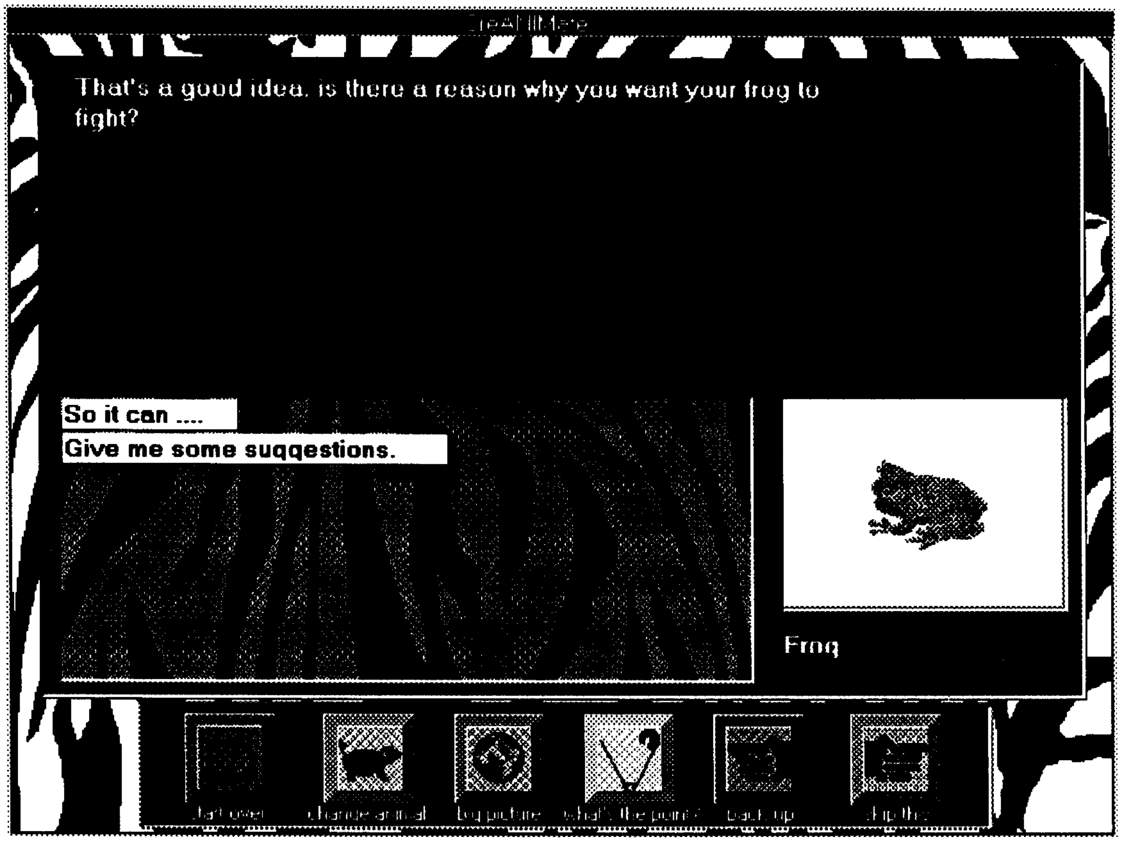

Figure 6. Posing a question: "That's a good idea, is there a reason why you want your frog to fight?"

In this case, the program asks the question, "Why fight?" after determining that its video library includes clips showing reasons that animals fight. The questions that the system asks are designed to teach the basic relationships that underlie animal adaptations. This particular question emphasizes the fact that animals do not engage in behaviors unless they help the animal to survive.

STUDENT: So it can ...

Proposing an answer to a question

When a student selects an answer containing an ellipsis, he is prompted to type in his own response in place of the ellipsis. The original options are replaced by a type-in window showing the partial sentence he selected and a blank for him to fill in.

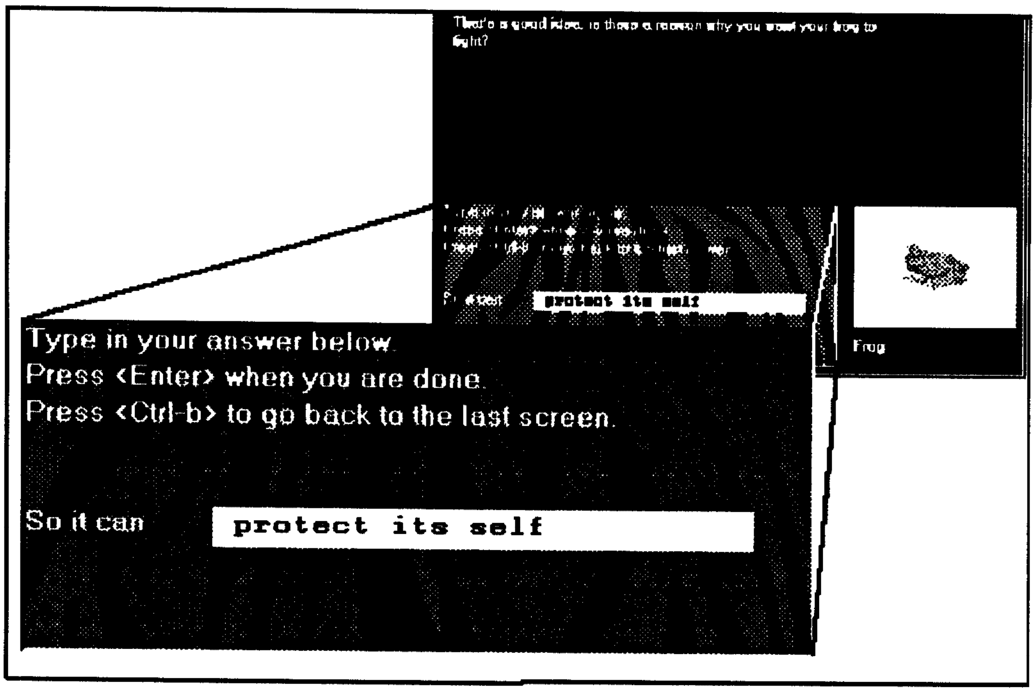

Figure 7. Typing in an answer to a question: "So it can protect its self."

STUDENT: So it can protect its self.

The ability to enter their own answers gives students an opportunity to generate their own hypotheses. While understanding free text is still an unsolved problem in artificial intelligence, we have reduced the problem to a manageable level by providing students with partial sentences to complete. In this case, the student's input contains a misspelling, "protect *its self.*" The Creanimate parser is able to accommodate some misspellings and errors in grammar.

Example Cases

Creanimate recognizes that protecting oneself *is* a reason that animals fight. Furthermore, it has video clips that show animals fighting to protect themselves. The student is offered the opportunity to see a video clip about bees that confirms his answer. Whenever Creanimate offers a video clip, it displays a picture of the animal from the video in the lower righthand portion of the screen.

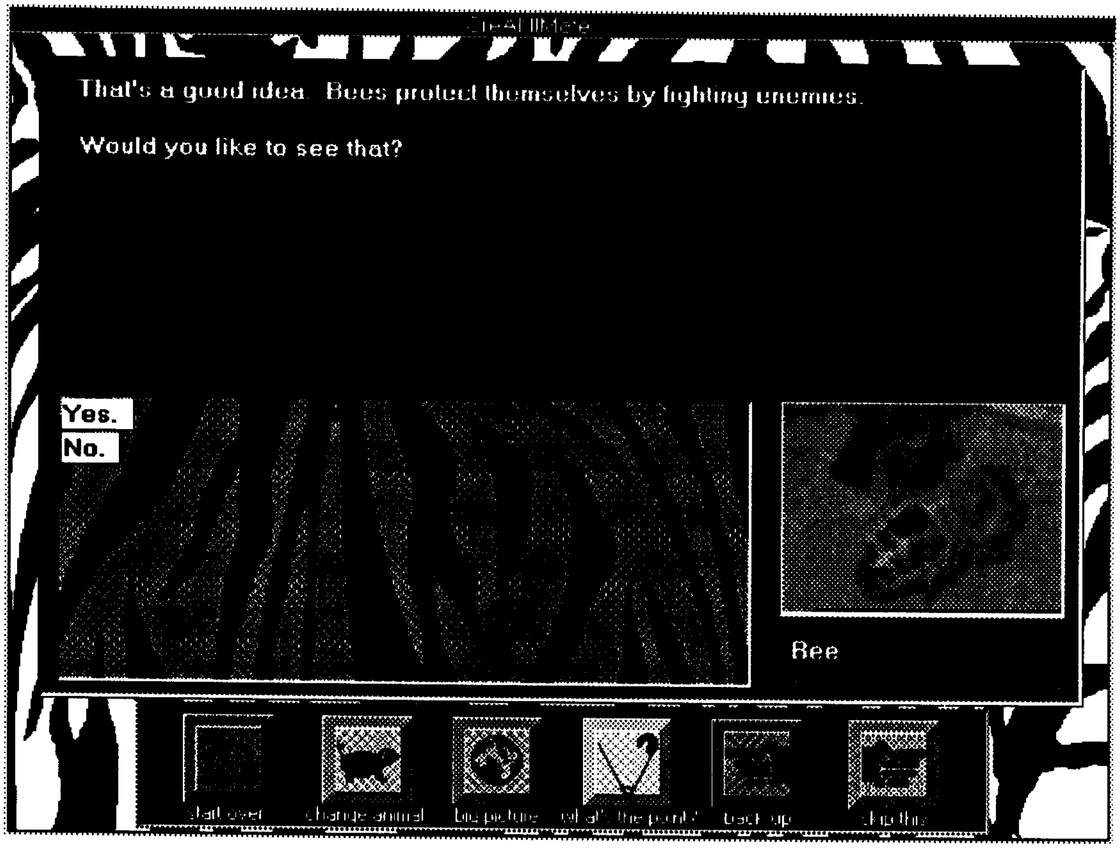

Figure 8. Introducing a video: "That's a good idea. Bees protect themselves by fighting enemies. Would you like to see that?"

STUDENT: Yes.

Seeing A Story

Displayed directly on his computer screen, the student sees a video about bees fighting a marauding wasp.

Action: A wasp is seen in the hive surrounded by bees. The bees swarm all over the wasp, stinging it repeatedly. Eventually, they drag the wasp out of the hive. It is left struggling to move.

Narration: *Not infrequently wasps invade the hive...*

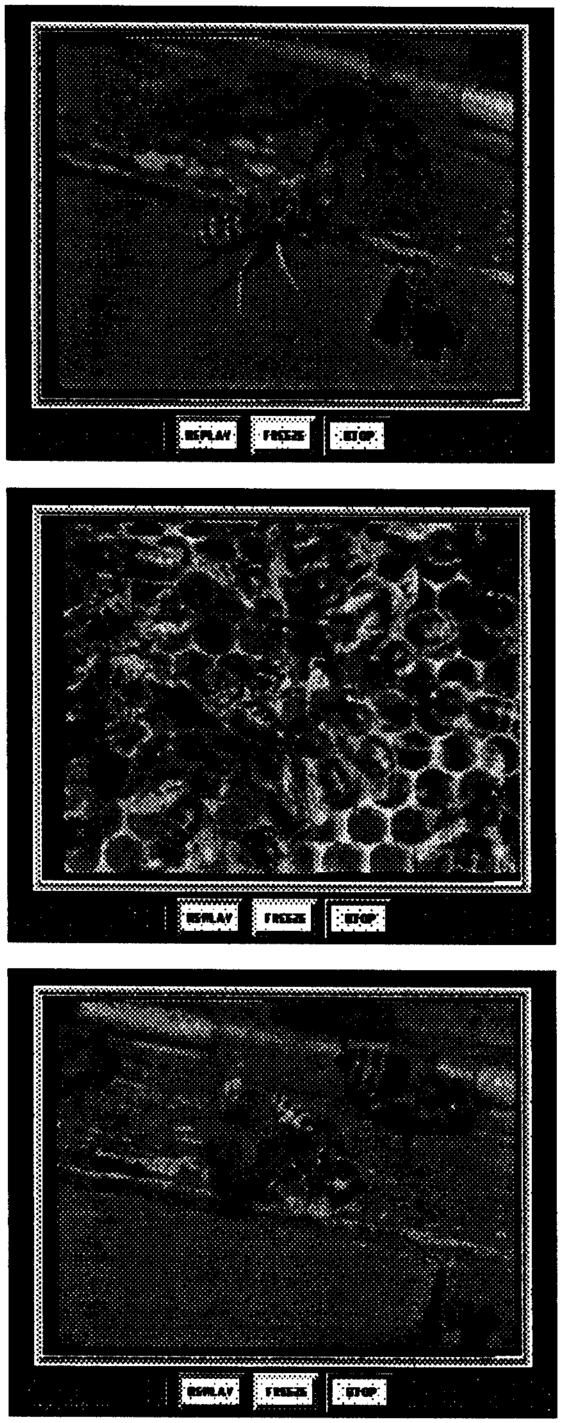

Figure 9. Seeing a video story

Additional Examples

After showing the preceding story about bees, Creanimate suggests a second one about a very different animal doing the same thing. This story also shows an animal fighting to defend itself. However, this one shows one jawfish defending itself against another. A jawfish is a fierce looking fish that hides in holes on the sea floor and darts out to catch prey in its oversized mouth. The diverse examples help to show students commonalities across the wide spectrum of animals.

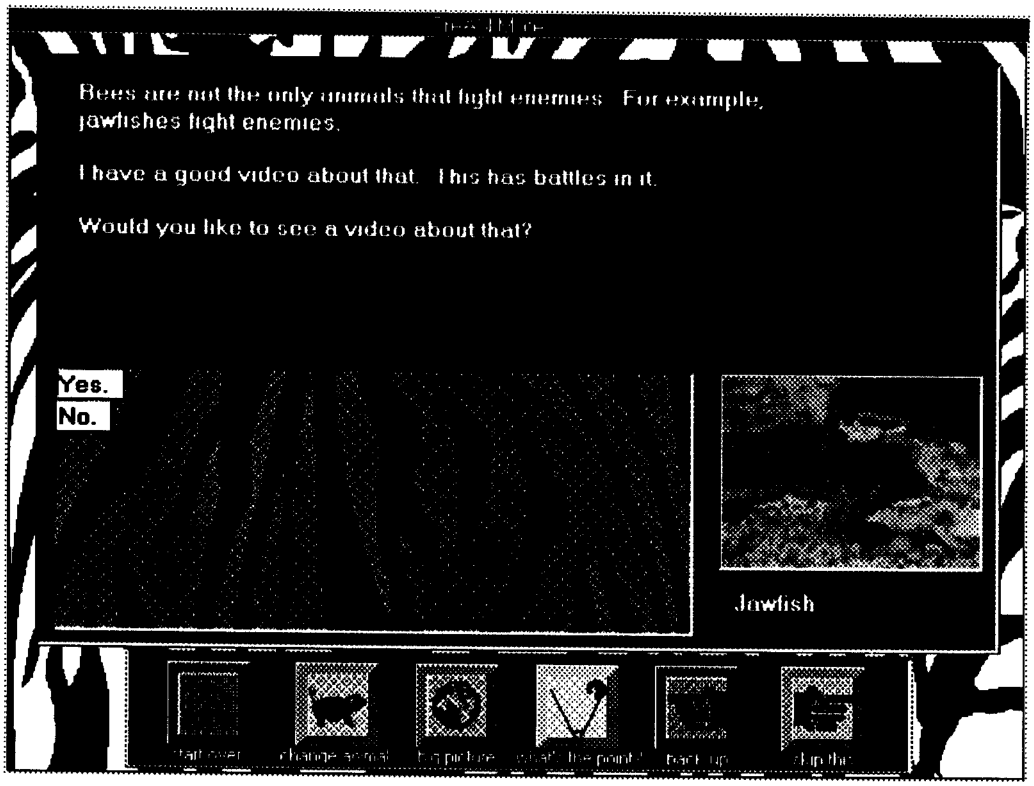

Figure 10. An additional example

STUDENT: Yes.

Committing to an answer

Once the student has seen examples of animals fighting to protect themselves, he has the opportunity to commit to that answer. If he commits, he goes on to other questions about his frog. Otherwise, he can consider additional answers to the current question.

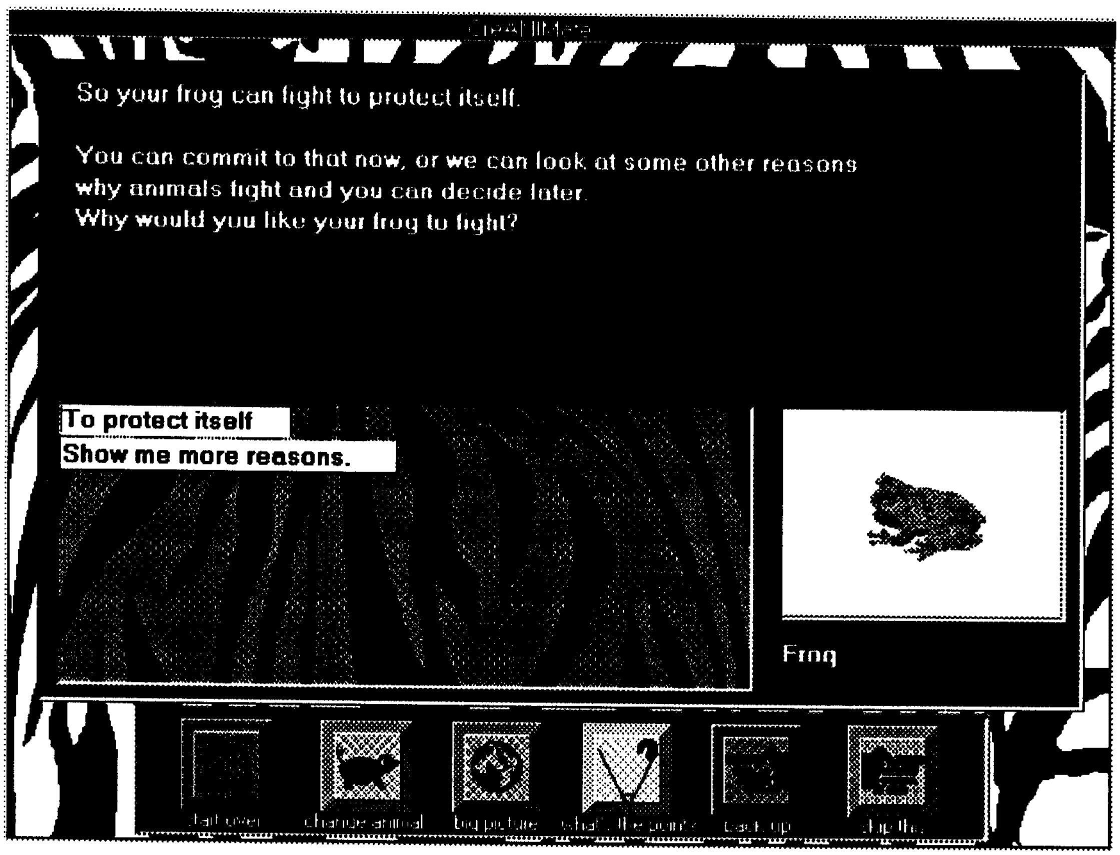

Figure 11. A commitment point: "So your frog can fight to protect itself. You can commit to that now, or we can look at some other reasons why animals fight and you can decide later."

STUDENT: To protect itself.

A Second Question

Now that the student has decided why his frog will fight, the program raises a new question: *How* will it fight? This new question emphasizes the fact that there are many different strategies pursued by animals in the course of achieving the same goal. In this particular case, there are several ways animals can fight.

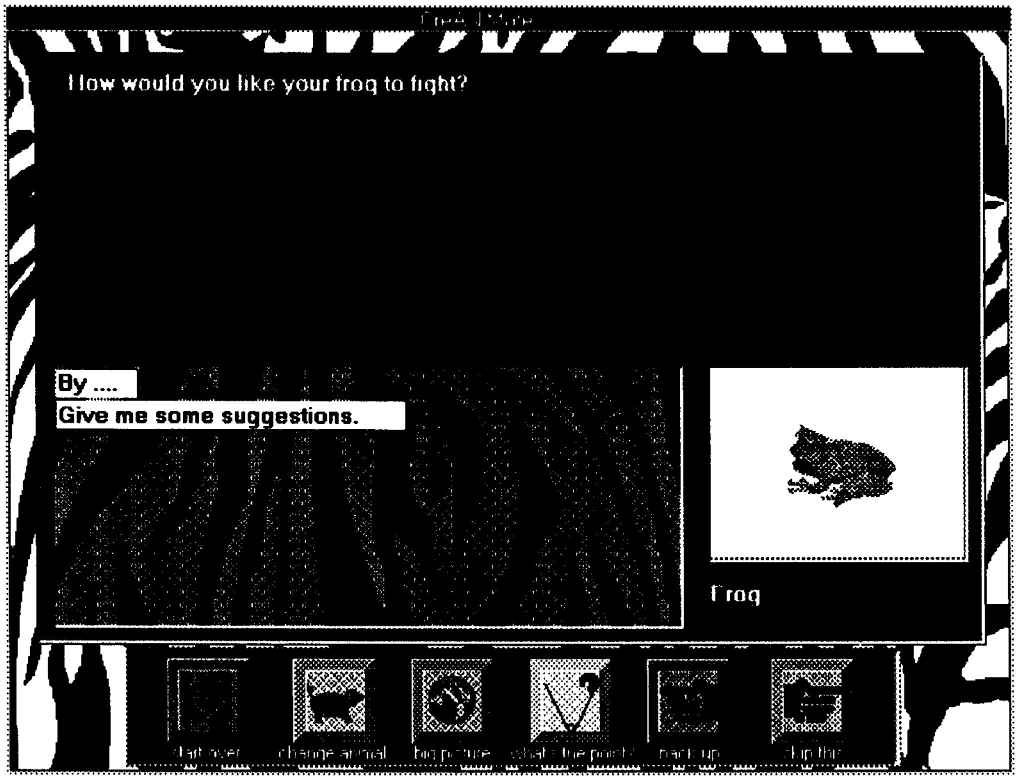

Figure 12. A second question: "How would you like your frog to fight?"

`STUDENT: By jumping up on him.`

The student selects the partial sentence, "By..." and types "jumping up on him." Creanimate does not have the action "jumping on an animal" in its knowledge base, so it reads the student's input simply as "jumping". The partial sentence "By ..." helps the student to understand what type of answer the system is looking for. In this instance, the student can tell that an action is an appropriate answer, but a physical feature is not. Within that constraint, the student has the full range to express himself.

Cases for Incorrect Answers

Creanimate does not know of any animals that jump in order to fight. In the absence of this knowledge, it takes the opportunity to show the student some reasons it knows for why animals do jump. In this way, students learn from incorrect answers as well as correct ones.

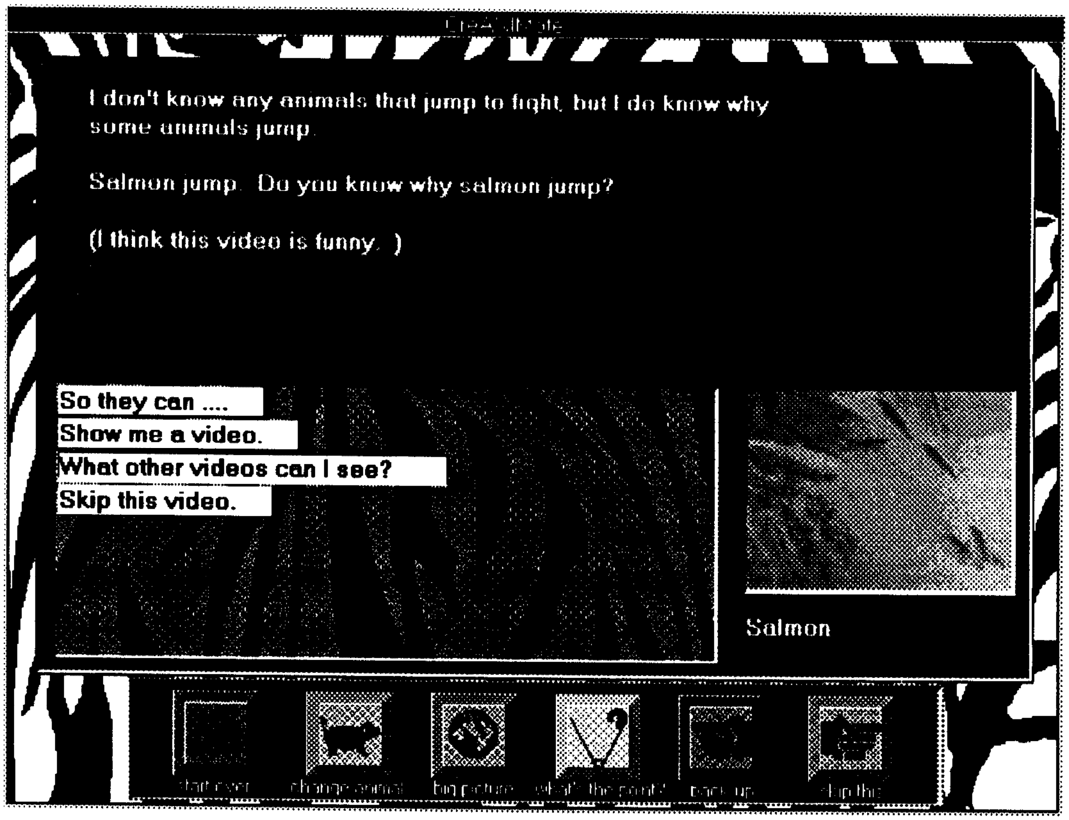

Figure 13. An example for an incorrect answer: "I don't know any animals that jump to fight, but I do know why some animals jump. Salmon jump. Do you know why salmon jump?"

Creanimate introduces the story about salmon with a question. This is a strategy to make students active viewers of stories. As a result of the question, students watch the video clip looking for the answer. It also helps to focus students on the aspects of the video that are relevant to the question under consideration.

```
STUDENT:    What other videos can I see?
```

Choosing a story from a menu

Instead of watching the salmon video, the student asks what other videos he can see. In response, Creanimate displays pictures of four animals labeled with the animals' names. Each of these corresponds to a story that shows a reason that animals jump. The student selects a video clip by clicking on a picture.

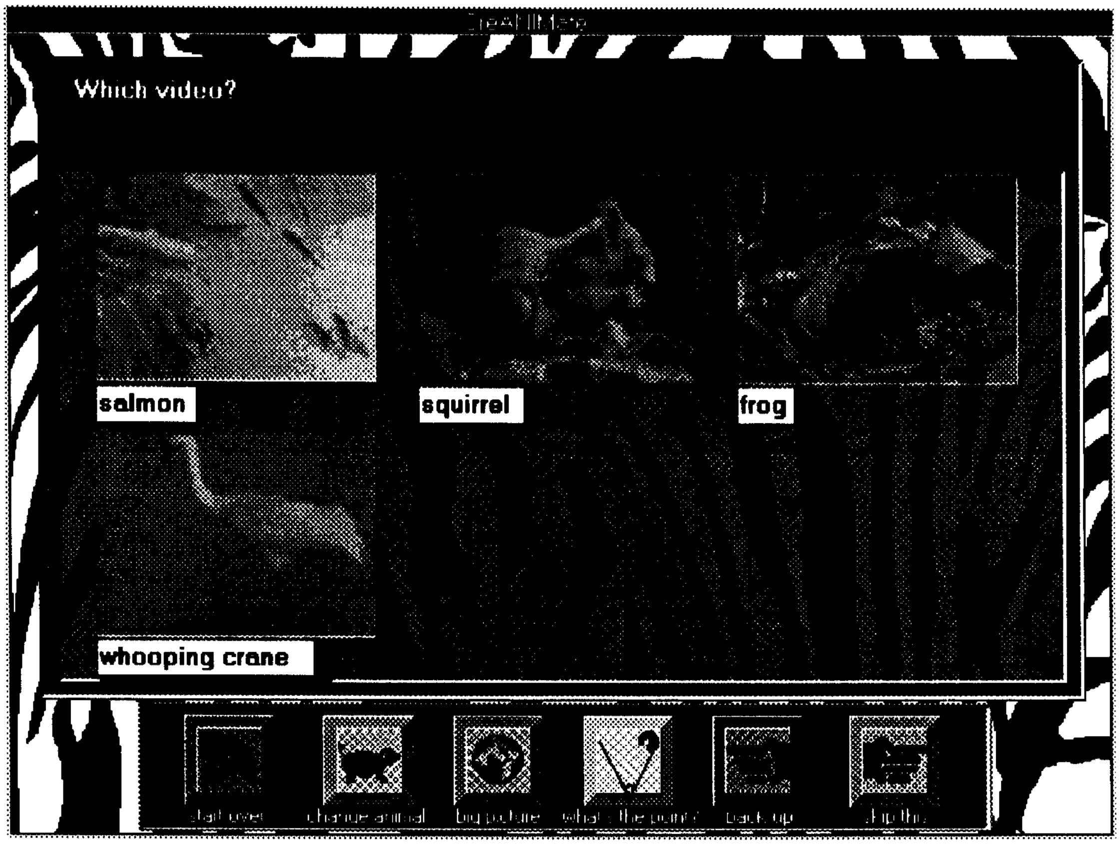

Figure 14. A menu of videos: "Which video?"

The student selects the picture of the whooping crane.

```
STUDENT:    Whooping  crane.
```

Advertising a story

To help "advertise" them, stories are labeled in the system's memory with information about what makes the particular video appealing. In the case of the whooping crane video, the story has been labeled with the attribute, "frantic dancing".

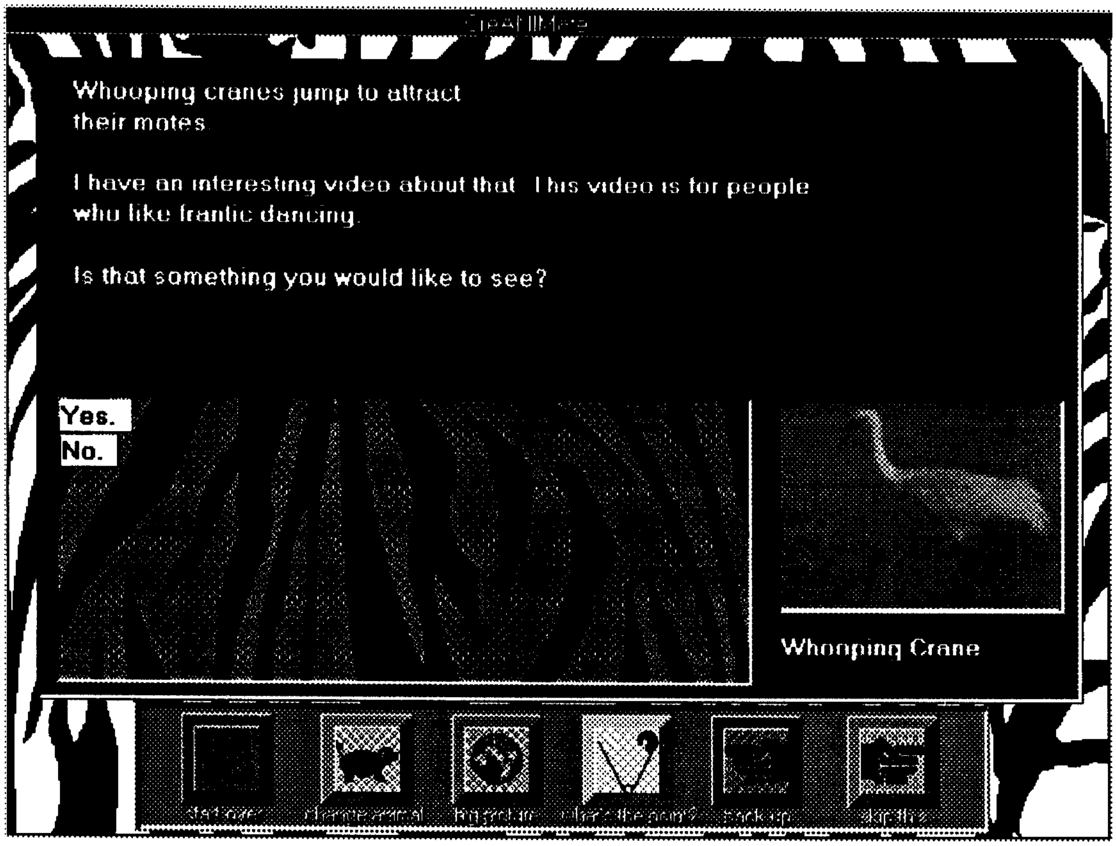

Figure 15. Advertising a story: "Whooping cranes jump to attract their mates. I have an interesting video about that. This video is for people who like frantic dancing."

STUDENT: Yes.

This video clip shows cranes dancing to attract their mates. The dance is an elaborate ritual consisting of jumping, flapping wings, and stretching necks.

Choosing where to go next

After seeing one reason for jumping, the student is given the choice between returning to the main issue, how his frog will fight, or learning more about jumping. Creanimate gives the student as much control as possible over his interaction. In this case, he gets to choose where the discussion will go next. He also may use the control buttons at the bottom of the screen at any time to back up, skip ahead, get some assistance, or change to a new animal.

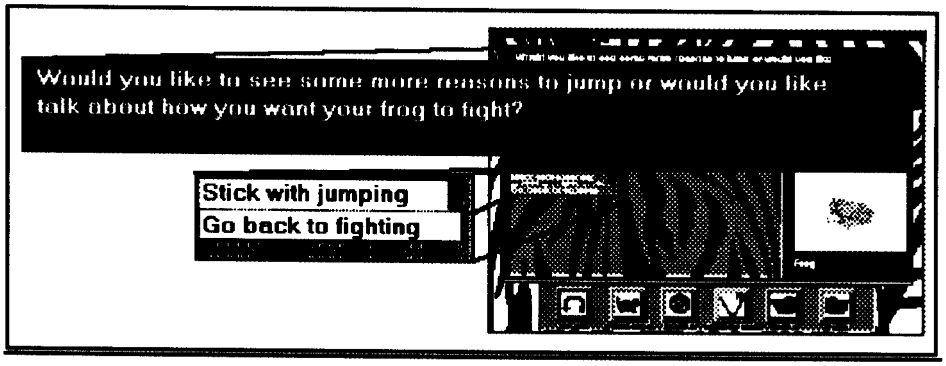

Figure 16. Choosing where to go next: "Would you like to see some more reasons to jump or would you like to talk about how you want your frog to fight?"

STUDENT: Stick with jumping.

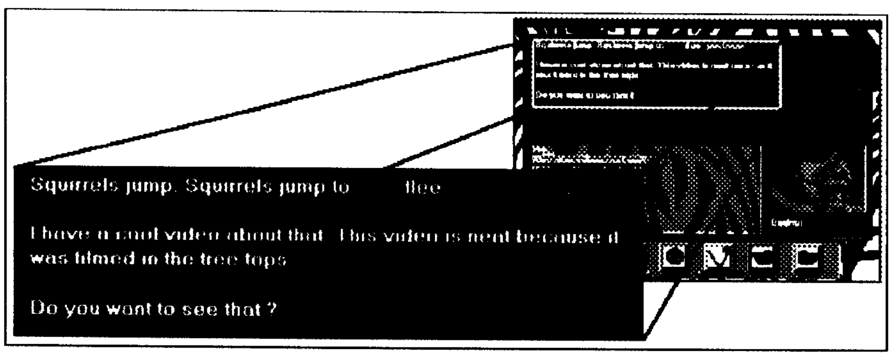

Figure 17. Another story about jumping: "Squirrels jump. Squirrels jump to flee predators." The word "predator" appears in yellow on this screen to indicate that it has a definition associated with it. If a student clicks on a yellow word, the program displays its definition.

STUDENT: Yes.

Suggesting an Answer with a Story

The student sees several more videos showing reasons animals jump before Creanimate runs out of them. It then returns to the main question: How will his frog fight? Creanimate suggests a way to fight using a story about bears fighting each other as an example case.

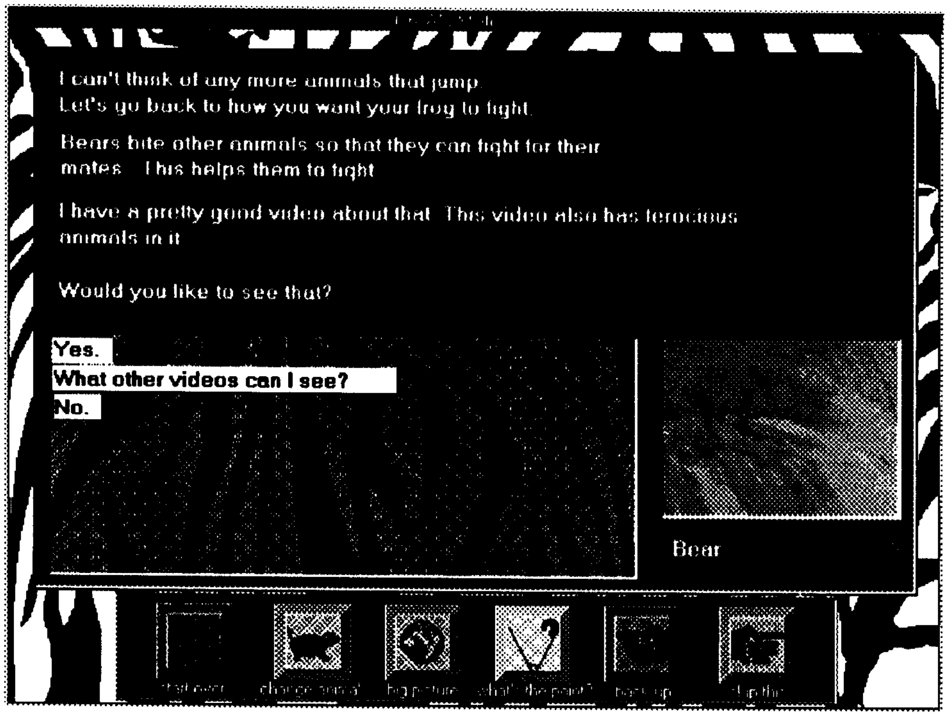

Figure 18. Suggesting an answer: "Bears bite so that they can fight for their mates. This helps them to fight."

STUDENT: Yes.

This video shows two male bears fighting each other. They growl ferociously, rear up on their hind legs, bite at each other, and slash with their giant paws. Accompanied by dramatic music, this is a riveting video.

Resolving the Question

At this point the student has discussed two ways for his frog to fight. He suggested jumping on other animals, but Creanimate did not know of any animals that fight that way. Instead, it suggested biting. So as not to inhibit creativity, the program will still allow him to commit to jumping if he prefers. He can also see more ways to fight before he commits.

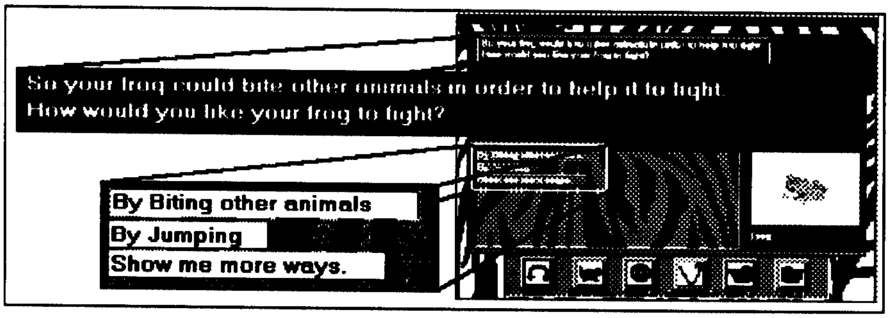

Figure 19. A commitment point for the second question: "So your frog could bite other animals in order to fight. How would you like your frog to fight?"

STUDENT: By biting other animals.

Examining Ramifications of an Answer

The student's resolution of the question *how to fight* raises a new question. What will his frog need to enable it to bite at another animal? By pursuing these chains of questions, students learn to analyze the ramifications of decisions beyond their immediate impact.

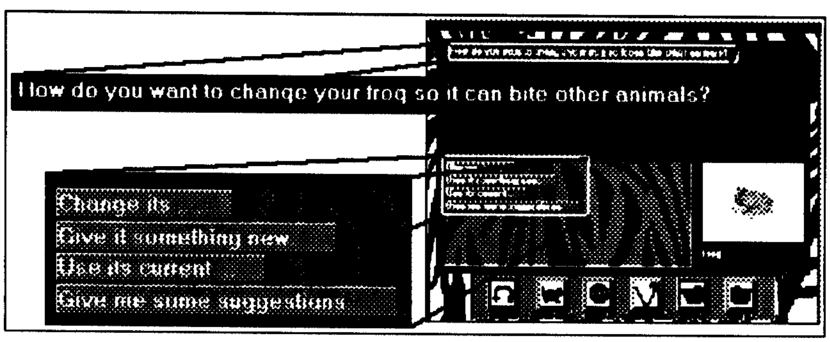

Figure 20. A new question: "How do you want to change your frog so it can bite other animals?

STUDENT: Skip this.

Student Control

At this stage, the student chooses not to pursue this issue. Instead of using one of the options provided to answer the question, he clicks on the "Skip this" button from the button pad at the bottom of the screen.

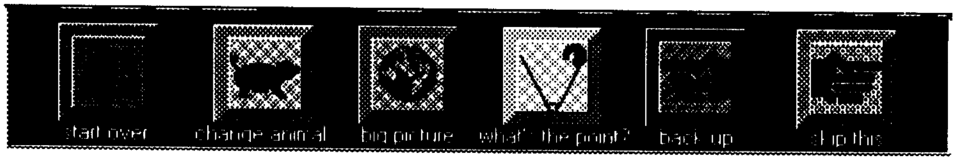

Figure 21. The Creanimate *User Control* button pad.

In response to this directive, the student is given several choices of what to do next.

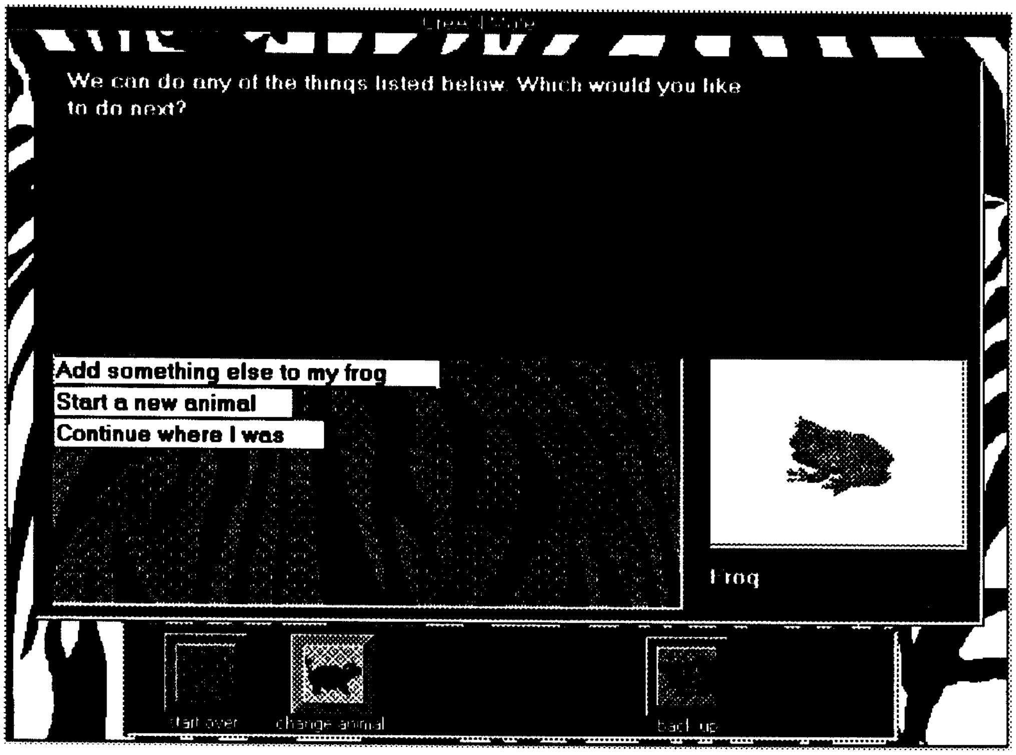

Figure 22. *Skip this* options. Depending on the context, the options that the student sees for "skip this" vary.

STUDENT: Add something else to my frog.

The "After" Picture

The student in this transcript chose to go on to make his frog hunt. However, if he had stuck with the current question he might have chosen to give it a beak to help it bite other animals. For a limited number of physical modifications, Creanimate can show the student a picture of his modified animal. It does so, only if the student has committed to both a new feature and a reason for the new feature. The student can reach this point by selecting a feature and answering the question about how the animal will use that feature or by selecting an action and adding the feature to enable his animal to perform that action. In the current transcript, the student had already selected the action *bite other animals* and the next step would have been to commit to a feature to enable it to bite other animals.

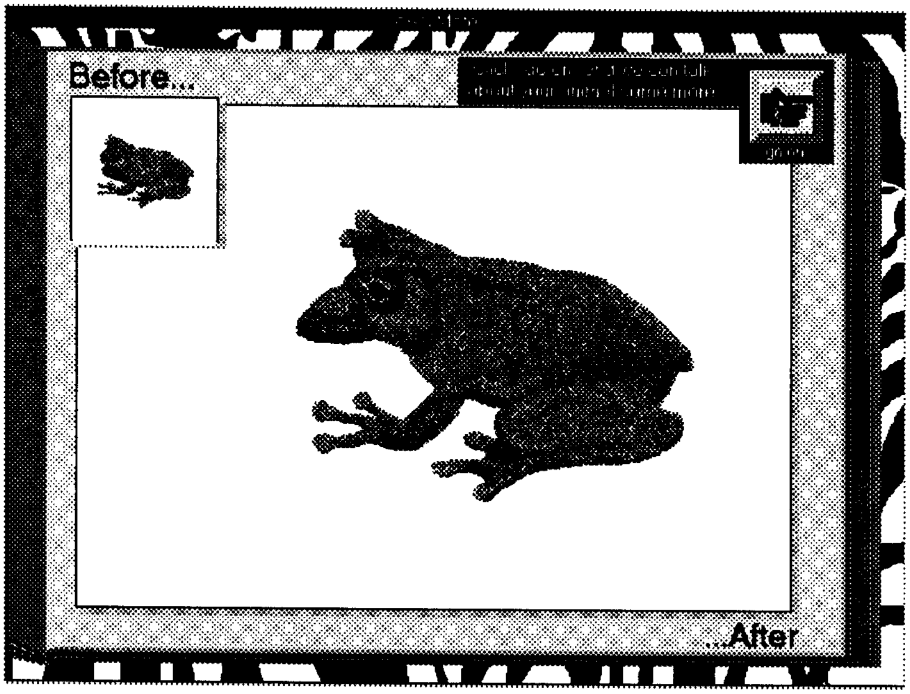

Figure 23. An "after" picture. A frog with a beak.

The "after" screen resembles the "before" screen only the picture of the unmodified animal has been shrunk and the question mark in the after box has been replaced with the student's new animal.

As the preceding transcript shows, the student takes the initiative in the learning interaction. A student initiates the dialogue with a proposal for a new animal. Creanimate responds to this proposal with an open-ended question which the student can either answer or ask for suggestions. When he is ready, he can commit to an answer. The questions that they consider teach students the important relationships that underlie the study of animal adaptation, and the video clips provide concrete examples. The structure provided by the questions and the cases provided by the video reinforce each other in the students' growing understanding of the domain.

What transcripts and descriptions can never convey, of course, is the drama of the stories that Creanimate presents. The videos of animals in the wild in Creanimate are a prime example of the superiority of video over less dynamic media. Children and adults alike find these video clips riveting. The challenge of constructing multimedia systems like Creanimate is placing this powerful medium in a meaningful context for learning. Anyone can collect exciting video and display it on demand. The real challenge is presenting it in a context that helps students to learn valuable and lasting lessons from it. In Creanimate, the questions concerning students' animals establish this context for learning. With Creanimate, students do not receive information in the predetermined, inflexible order of traditional, linear video, nor do they see it in a detached context in which they have no motivation for learning from it. Instead, they see it in the context of a discussion in which they are invested, in response to questions that encourage them to construct and explore their own hypotheses. The discussions that students have with Creanimate provide them with both a motivation to acquire new knowledge and perspective from which to integrate it. The vividness of the medium helps them to retain it.

Creanimate teaches Questions and Cases

Expertise is often viewed as the possession of a large body of knowledge. However, knowledge of facts is only one component of expertise. What sets an expert apart from a novice is the ability to deal effectively with new situations within his realm of expertise. When confronted with a novel situation, an expert knows the right questions to ask and how to go about answering them. The cases and facts that the expert commands help him or her to resolve the questions, but the questions come first. The basic questions that an expert uses to explain phenomena in a domain are called *explanation questions* (Schank 1986). Creanimate provides students with the tools for expertise in the domain of animal adaptation. It teaches the questions to ask when confronted with new situations, and it provides students with the cases that go along with these explanation questions. It helps students develop the skills necessary for case-based reasoning in a scientific discipline.

Status and Evaluation of Creanimate

Creanimate is a working program that has been successfully tested in a variety of environments. The most recent version contains 140 indexed video clips totaling 60 minutes. It conducts dialogues about six different questions, each focusing on a different relationship between the physical features, actions, and behaviors of animals. The knowledge base includes over 200 animals and over 1000 different characteristics of animals. Included in this knowledge base is information that allows Creanimate to construct natural English sentences using the concepts in its memory and to understand typed input by students. Our experience with the program indicates that students can use the current prototype for at least three hours without exhausting its capabilities.

The principal tests have been in two Chicago-area public schools. In the first test site, Creanimate was tested with 24 fourth graders and in the second, it was tested with 15 students in fifth through seventh grades. In both cases, the testing had several objectives. The primary objective was to evaluate student response to the program. In both schools, students responded enthusiastically. Each student used the program for three sessions of approximately 45 minutes. In surveys, students rated the system highly. Specifically, they indicated a preference for Creanimate over both ordinary science learning and watching nature shows on TV. However, since students knew their surveys were going to be read by the system's developers, we felt it was necessary to look for more accurate reflections of how they felt. These indications came both from direct observation and feedback from their teachers. For example, in nearly every case, students had to be told several times that their session was over and that it was time to leave. "Just one more minute?" was a common response. In addition, students were always asked whether they wanted to return for the next session, and none ever said no. Teachers at the first site reported that their students were upset when they found out they would not be able to continue to use the program when they returned for fifth grade in the fall. In addition to gauging student response, we also developed assessments designed to measure learning. Specifically, we were looking for changes in students' question-asking and explanation generation as a result of using Creanimate. In the pre- and post-tests we designed, students were given pictures of exotic animals and asked to either generate questions about the unfamiliar animals or answer questions about them. These tasks were chosen for assessment because they correspond to the authentic practice of naturalists and biologists. The results of these evaluations are reported in Cleave, Edelson, and Beckwith (1993).

The conclusions from our experiences with Creanimate in the schools have been positive. Students remained focussed on the program throughout their sessions. Their overt reactions to the video clips ranged from laughter to admiration to revulsion. They could be seen thinking carefully about the animals they created and seriously deliberating over their answers to the program's open-ended questions.

Creanimate provides an opportunity for students to engage in the creative and imaginative practice of science that is missing from most current education. It does so by bringing these processes to a level that connects with children's fascination with both animals and fantasy. Starting from this point it takes them through a scientific inquiry that exposes them to the important relationships that underlie the study of animals in the wild, and to vivid cases that demonstrate those relationships.

1.3 The Case-based Teaching Architecture

Creanimate is a prototype for a new class of learning environments called case-based teaching systems. The case-based teaching architecture is designed around two basic principles: *active learning* and *learning from stories*. Active learning demands that a student be actively engaged in a meaningful task in the course of learning. A meaningful task provides both the motivation and context for learning. The principle of learning from stories acknowledges that a great deal of learning, both formal and informal, comes from hearing stories. Good parents and teachers all tell stories at opportune moments to teach important lessons. Teaching with stories is a natural way to support the important process of case-based reasoning. Teaching with stories relies on the possession of two critical skills: the ability to recognize an opportunity to teach with a story and the ability to identify the right story to tell in a particular situation. Thus, to teach with stories one must know how to tell the right story at the right time.

A case-based teaching system supports active learning and learning from stories through two interdependent components, the *task environment* and the *storyteller*. The task environment provides a student with an engaging, motivating task. The storyteller monitors the task environment looking for opportunities to tell stories that will help the student to learn from his interactions with the task environment. In Creanimate, the task environment consists of a question-and-answer dialogue, and the storyteller uses a video library of animals in the wild. A task environment must be constructed so that it exposes students to an appropriate variety of situations. These situations must give them sufficient experience of the domain. In his interactions with the task environment, whenever a student is in a situation that constitutes an opportunity for learning, the storyteller must be able to intervene with a story that will help the student learn from that situation. Therefore, the storyteller must have a sufficiently wide range of stories to cover the situations for learning that may arise in the task environment. In addition, these stories must be indexed in such a way that based on its observations of the task environment, the storyteller can locate appropriate stories for the situations that arise. In this architecture, the task environment teaches a student the underlying structure of the domain and the storyteller provides him with cases to support that structure. In practice, the construction of case-based teaching systems is both an art and a science. Creating engaging task environments that are motivating, challenging, and interesting is very much an art, while indexing stories so they can be retrieved at the right time is a science.

Indexing and Reminding in Case-based Teaching

When a person thinks of a story to tell, we say that he or she is reminded of a story. We usually think of reminding as a passive process, something that happens to someone, not something a person does. Upon examination of the process though, we find a set of processes and representations devoted to actively extracting features from the world and using those features as cues for the retrieval of useful cases or stories (Schank et al. 1990a; Schank et al. 1990b). The processes are called reminding strategies and the representations are called indices. An index is a label for a case that allows a reminding strategy to recognize situations in which the case is relevant. Since teaching is an expertise, teachers have a particular set of strategies that enable them to observe their students and retrieve appropriate cases. Two core issues for the development of effective case-based teaching systems are 1) the identification of strategies for reminding to achieve educational goals and 2) the development of algorithms and indexing schemes to implement these reminding strategies. Creanimate employs several reminding strategies that each enable the system to achieve specific pedagogical objectives. Each strategy places particular demands on the information available in the indices that label stories in the program's memory. Creanimate's reminding strategies are example remindings, similarity-based remindings and expectation-violation remindings.

Example Remindings. The bread and butter reminding for the CreANIMate system, just as it is for any teacher, is the example. For instance, in a dialogue in which the student asked for a tortoise that could run fast, Creanimate posed the question, ``What should we give your tortoise so it can run fast?" When the student asked for suggestions, the program responded:

```
Cheetahs run fast.  Do you know what cheetahs have to help
them run fast? (I have an awesome video about that.)
Student:    They have   long   legs.
That's right.  Cheetahs have long, muscular legs to help them
to run fast.  Would you like to see that?
```

Performing example remindings requires that specific information be available in an index. An index must describe the questions and answers for which the story can be used as an example. The relationship between physical features and the functions they support is one of the central lessons of the system. Therefore, part of an index may indicate that the story demonstrates the use of a particular physical feature, e.g., *long, muscular legs*, for a particular reason, *to run fast*. For every feature/function pair that appears in a video clip, there is a corresponding entry in the index indicating the presence of that pair in the story. The same is true for actions that are used to achieve a survival behavior, for instance, *to run fast to pursue prey*. Each one of these pairs can be used to answer two questions. The story in the example above could be used to illustrate either, "Why have long legs?" or "How to make an animal run fast?"

Similarity-based Remindings. One of the risks of teaching with examples is that students may draw overly specific conclusions from the examples that they see. Therefore, one objective of a case-based teacher is to assist the student in making generalizations. The strategy that Creanimate uses to help the student form appropriate generalizations is called similarity-based reminding. In similarity-based reminding, the system retrieves a story that illustrates the same basic principle as a previous example, but is sufficiently different to allow the student to form a generalization at an appropriate level of abstraction. The following example of a similarity-based reminding was initiated by a student's request for a tortoise that could run fast:

```
Cheetahs run fast. Do you know why cheetahs run fast? (I
have an impressive video about that. )
```

Student: So they can catch other animals.

```
That is right. Would you like to see that?
```

Student: yes

[VIDEO: Cheetah Pursuing Prey]

Similarity-based reminding:

```
That reminds me of a cool video. Fishing bats also move fast
in order to get food. Only, instead of running fast to
pursue their prey, they fly to pounce on their prey. Would
you like to see that?
```

STUDENT:yes

[VIDEO: Fishing Bat]

In this example, the program presented a video of a cheetah that runs fast to pursue its prey. The cheetah story was produced as an example of a reason that animals run fast. However, to ward off the possibility of the student drawing an overly-specific conclusion, the program presents a similar story about an animal that moves fast to get its food, but instead of running fast, it flies. In order to perform similarity-based reminding, the system must be able to identify stories that are similar, but not identical, to the given example. This is done by searching through its knowledge base looking for videos that share an abstraction. In this case, both stories share the abstraction: *move fast to hunt*. Similarity-based remindings also serve the important purpose of exposing students to things they might not see otherwise, thus broadening their awareness and increasing their curiosity.

Expectation-violation Remindings. Surprise is a great motivator for learning. When some expectation that you have is violated by an experience or observation, you become surprised and are motivated to understand why your expectation was not met. This is what Schank (1982) calls failure-driven learning. In this case, failure refers to the failure of an expectation to explain an observation, not the failure of an individual to achieve a goal. Expectation failures do not just promote learning they provoke interest. The reminding strategy called expectation-violation reminding capitalizes on surprise to provoke interest on the part of a student and broaden his exposure. To present videos that violate a student's expectations, the system must have knowledge of what sorts of things students are likely to believe. This information is added to the knowledge base when stories are entered. Expectations take the form of rules, such as "Mammals do not fly." An expectation of this form leads to the following expectation-violation reminding:

```
If you could create a new animal by taking an existing animal
```

```
and changing it some way, what would you make?
Student:    A dog that can fly
Actually some mammals do fly.  For example, fishing bats fly.
I have a cool video about that.  Would you like to see that?
Student:    yes
[VIDEO: Fishing Bat]
```

Part of the index for the fishing bat story contains the information that it violates the expectation that mammals do not fly. When the student asks for a mammal that flies (in this case, a dog), this matches the expectation-violation in the story and the program gets reminded of that story.

All three reminding strategies capitalize on a student's current context to present a story that is directly relevant at that moment. The cases that a student sees are in response to an action or answer that he has made in the course of his session. The responsiveness of this approach helps maximize students' abilities to be active learners and to learn from stories. The stories retrieved by the reminding strategies provide concrete cases to illustrate the current discussion, promote appropriate generalizations, and widen a student's exposure. Each of these, in turn, promotes curiosity and additional inquiry.

1.4 About this Dissertation

In the first section of this chapter, I described an intentionally ambitious research program. It is important to distinguish between the portion of that research program described in this dissertation and the research program as a whole. This dissertation covers one step of that program. In particular, the research described here was devoted to the development of a theory of indexing and reminding for teaching and the implementation of that theory in an AI system. The theory is expressed in the form of the case-based teaching architecture and in the set of reminding strategies described here. It extends theories developed by Schank (1990; 1991) from an abstract characterization of the role stories and cases can play in teaching to a specific architecture and a set of reminding strategies for teaching with stories. The specific contributions of this work are:

- Identifying a Socratic-style dialogue as a means for establishing a context for teaching with stories.
- Developing and implementing an architecture for a computer-based dialogue manager that conducts a question-and-answer dialogue that establishes a context for learning from stories.
- Identifying specific reminding strategies that capitalize on opportunities to teach with stories in the course of a Socratic-style dialogue.
- Developing an indexing scheme for a story library and appropriate algorithms for implementing these reminding strategies.

The role of the Creanimate system in this research is to explore the technical issues involved in this new architecture, to demonstrate the ability to construct such a system, and to provide an initial platform for investigating its educational effectiveness. Creanimate is not yet a complete system, nor has it satisfied all of its goals. However, it does have important value as a demonstration of an architecture for teaching with stories in the context of a Socratic dialogue. Any faults in Creanimate's particular design or educational effectiveness must be considered separately from the value of its architecture as a generally effective teaching architecture.

Creanimate is the first step in this research program. It is the embodiment of an AI theory of indexing and reminding in the service of teaching. The essential next step in this research program will be to evaluate the educational effectiveness of that architecture. This dissertation outlines a theory of of learning that explains why this architecture would be effective (Chapters 2 and 3), but this theory has yet to be empirically verified. Creanimate by itself will be insufficient for evaluating the effectiveness of this architecture. Separating the strengths and weaknesses of Creanimate from the strengths and weaknesses of the architecture itself will require systems in different domains aimed at students at different levels. The evaluations of Creanimate that I described above and in Chapter 9 should be viewed as preliminary, intended only to gain early experience that can inform future development and testing.

This dissertation is divided between a discussion of the educational objectives and theories that underlie the teaching architecture described here and a description of the architecture itself. The architecture has been investigated through the development of the Creanimate system, but the two should not be confused. Creanimate is one of many possible implementations of that architecture.

In the next two chapters, I cover the educational aspects of the Socratic case-based teaching architecture. In Chapter 2, I present a theory that justifies teaching with stories and questions. This theory of teaching and learning is grounded in the theory of case-based reasoning. In Chapter 3, I describe the case-based teaching architecture in detail. In that discussion, I present factors for effective case-based teaching system, and I describe how they apply to Creanimate.

Chapters 4 through 8 are devoted to the implementation of case-based teaching from an AI standpoint. Because the reminding process is central to case-based teaching, it is the subject of Chapters 4 and 5. In Chapter 4 I describe the reminding strategies that Creanimate employs, and in Chapter 5 I present the algorithms that were used to implement them. In a case-based teaching system, the reminding strategies take advantage of the context for learning established by the task environment. In the Socratic case-based teaching architecture, the task environment is managed by a dialogue manager that engages the student in a question-and-answer dialogue. This Socratic dialogue manager is described in Chapter 6.

Supporting both the reminding strategies and the dialogue manager in Creanimate is an elaborate knowledge base that encodes domain-specific information about animals and their adaptations. In Creanimate, the knowledge base provides both a vocabulary for indexing stories and an information source for drawing the inferences required by the dialogue manager. I describe this knowledge representation in Chapter 7.

In Chapter 8, I describe the way the knowledge representation that is presented in Chapter 7 is used to construct the indices that label stories. These indices enable the storyteller to retrieve the right story at the right time. Chapter 8 describes the indices themselves, the role of the human indexer in constructing these indices, and the Creanimate Indexing Tool, a software environment that we developed to assist the indexer with this process.

In the final chapter I provide two additional perspectives on the research described here. First, I discuss some related work in both education and artificial intelligence, and then I describe ways to extend this research program beyond its current state.

Chapter 2

Teaching with Stories and Questions

If you reflect back on the best teachers you've known, you will often find that they possess two qualities that make them exceptional. They ask good questions, and they tell good stories. The aim of this research is to develop an architecture that makes use of these two educational techniques in the same effective way that a talented human teacher does. Before moving on to an examination of how to construct a system that teaches with stories and questions, it is important to understand how they can be used in teaching, and why they are effective.

2.1 Teaching with Stories

Good parents and good teachers all use stories to convey lessons. Teaching with stories comes as naturally and unconsciously as learning from them. Many institutions of formal instruction, such as business schools, law schools, and medical schools, have been teaching with stories for decades although they call it teaching with cases, not stories. In fact, stories and cases are intimately related[1]. Traditionally, stories have been conveyed orally or in writing, but new technologies allow us to convey stories through film, graphics, and animation. In 1977, James V. McConnell was awarded the American Psychological Association's Distinguished Contribution to Education in Psychology Award for his extremely successful introductory college psychology textbook, *Understanding Human Behavior* (McConnell 1977). In his acceptance address, "Confessions of a Textbook Writer" (McConnell 1978), McConnell attributes the success of his textbook, in large part, to his use of stories. In this address he quotes the advice of his mentor, "If you want to capture the imaginations of young people, you have to tell them stories!" (McConnell 1978, 160) Never has that counsel been truer than in the modern world with dramatic, engaging media vying for students' attention. The key to capturing the imaginations of young people is telling stories that exploit the powerful storytelling media now available.

In this section, I deal with stories in teaching from four different perspectives. First, how are stories used in teaching? They are used either to establish an educational context or to provide a lesson in some context. Second, why is storytelling an effective method for teaching? The theory of case-based reasoning argues that people solve problems, construct

---

[1] In the terminology used here, a *case* is the form in which an experience is stored in a person's memory. A *story* is the form in which a case is communicated from one individual to another.

plans, and understand new situations by making reference to similar previous experiences, called cases. Hearing stories is one way that people enlarge their personal case libraries. Third, what makes the computer particularly well-suited to teaching with stories? The computer provides the capability to establish meaningful contexts for teaching with stories. In addition, computers offer the ability to store and present large libraries of stories that use vivid multimedia. Finally, what exactly constitutes a story, and how can a story be distinguished from other forms of teaching? Stories are specific depictions of characters and events that are conveyed as a form of communication.

2.1.1 Teaching with Stories: The Role of Context

Stories are employed two different ways in teaching, to create context and to respond to context. The former is the technique employed by the well-known case method of business and law schools, while the latter is what we refer to as case-based teaching. In storytelling to establish context, a story is presented that poses a problem or describes an important situation. Learning takes place through the exploration of the issues provided by the story. In storytelling that *responds* to context, stories are presented in the course of solving a problem or discussing issues. Stories that respond to context are presented at the moment that they are relevant to the instructional setting and they help a student to learn from this context.

Storytelling that Establishes Context

Consider the following story taken from *Psychology Today—An Introduction,* a psychology textbook for college students that James McConnell edited in 1969. The story introduces a chapter on physiological psychology :

> You are a 30-year-old housewife with a husband, two children and a fairly happy life. But about five years ago you began having severe headaches. The headaches turned into epileptic seizures, and eventually the seizures became so frequent that they were occurring a dozen times or more each day. You felt so bad that you wanted to die. Then one day the doctors told you about a new operation that might help. The surgery involved separating one half of your brain from the other. The doctors warned you that, even if the operation was successful, there might be rather peculiar side-effects. You were so desperate that you told them to go ahead. After the surgery, the epileptic seizures stopped almost entirely....You felt good for the first time in years. But then you noticed that your lefthand had started doing some very strange things...(McConnell 1978, 164)

This story establishes a context for the technical material that follows it. It provides a framework that the author uses to discuss the physiology of the brain and some specific split-brain research. The context that this story establishes is one for dealing with the issues concerning the locations of different functions in the various parts of the brain. The author has the following to say about his reasons for introducing the chapter with this story:

> ...As you can see, I was trying to present some fairly complex brain research from a point of view the introductory student could understand and identify with. I assumed that once the student got interested in the housewife, the student might be motivated to master the technicalities of the brain well enough to comprehend why the woman's lefthand was doing all those crazy things. (McConnell, 1978 #14, 164)

McConnell uses this story to provide a context for some highly technical material. He recognizes the usefulness of a concrete story that the students can relate to because the story gives the students a touchstone. As he says, the story gives the students something to "identify" with; it also gives them a concrete frame of reference they can return to as they digest the difficult material in the chapter.

The case method employed in law, business, and medical schools uses stories to establish context in a similar way. In graduate schools of business, for example, a case is distributed to the students that describes a problem situation for a real or hypothetical business. The students prepare solutions to the problem posed by the case. In class, the instructor conducts a discussion over the various solutions to the problem. In cases that are drawn from real events, the solution that was tried and its result will often be discussed too. In this teaching technique, the cases are in essence stories that establish a context for a discussion of important educational issues.

Storytelling that Responds to Context

Storytelling that responds to context is really the opposite of storytelling to establish context. Instead of providing a context for learning, this form of storytelling takes advantage of a context for learning as an opportunity to convey a lesson with a story. When someone uses a story to respond to context, he tells the story in response to a situation in which the story's lesson applies. The context influences both the telling and the understanding of the story. When a storyteller tells a story in a context, he relies on that context to help the hearer draw the appropriate lesson from the story. For example, consider the following story:

> When I was in 4th grade, there was a time when the top students would all rush to be the first one to finish a test. So, the teacher told us the following story:
>
> > Today on the way to work, the traffic was moving slowly. There was one guy who kept weaving in and out to try and beat the traffic. Finally, about ten minutes later, I looked over and noticed that he was right next to me even though he had been rushing as fast as he could.
>
> She told us that story so that we would realize that the most important thing was reaching our goal, not how quickly we did it.[2]

The story told by the teacher responds directly to the situation in the classroom. She told it at the moment the story would have maximum impact—when the hearers, like the story's protagonist, were rushing for no good purpose. She chose to tell it in a context in which the hearers were acting in a way that is opposite to the lesson of the story.

The preceding story, like a fable, carries a lesson that gives the listener guidance on how to behave. Stories told in context can also clarify an abstract technical point. For example, the following hypothetical story is used by Richard Feynman, the renowned physicist, to help clarify his discussion of a complicated issue in relativity. The story responds to its context by making an abstract idea concrete.

---

[2] This story and the subsequent one about flies and honey were collected by asking volunteers for examples of incidents in which they had been taught something with a story or had taught someone else something with a story.

> To continue our discussion of the Lorentz transformation and relativistic effects, we consider a famous so-called "paradox" of Peter and Paul, who are supposed to be twins, born at the same time. When they are old enough to drive a space ship, Paul flies away at very high speed. Because Peter, who is left on the ground, sees Paul going so fast, all of Paul's clocks appear to go slower, his heart beats go slower, his thoughts go slower, everything goes slower, from Peter's point of view. Of course, Paul notices nothing unusual, but if he travels around and about for a while and then comes back, he will be younger than Peter, the man on the ground! (Feynman, Leighton, and Sands 1963, 16-3)

Context does not just determine when a storyteller chooses to tell a story, it also influences how he or she may tell it. In the following story, a father tells his son a story in a context in which the child was acting unpleasant in order to get his way.

> Mark [our six year old son] has been demanding things and ordering other people around. He seems to be trying to discover if it's better to act this way, or to be pleasant. Of course, we prefer it when he's pleasant! So I told him a story based on the proverb "You catch more flies with honey than with vinegar." It went something like this:
>
>> Once there was a man who wanted to catch some flies. First, he poured out some stinky vinegar on the sidewalk and waited to see if any flies would come. He waited and waited, but none came. Then, he put some honey down instead. Suddenly, he had lots of flies! He realized that you can catch more flies with honey than with vinegar.
>>
>> Mark, when you demand things and shout at people to get things done, it's a lot like vinegar, kind of stinky. But when you're pleasant and cooperative, it's just like honey. It's very sweet, and we usually want to give you what you're asking for.
>
> This has been an effective story: we can say, "Mark, could you give us a little honey instead of vinegar?" and he usually stops and thinks about what he's doing, and often chooses to be pleasant.

The context in which the father tells this story to his son influences his telling of the story. He draws explicit links between the honey in the story and pleasant behavior on the son's part and between vinegar in the story and unpleasant behavior. These "bridging" comments are very important for helping the hearer to understand the lesson of a story and its relevance to the current setting.

The influence of context is not limited to the storyteller. Context also influences a hearer's interpretation of a story. A single story can convey several different lessons and can be told in different contexts to take advantage of those lessons. Schank (1990) describes an experiment in which graduate students were told stories out of context and then asked to respond to them the way they would if they heard them in a conversation. The subjects' responses were analyzed to see what points they extracted from the story they heard. The responses reveal a very wide range of interpretations that were influenced in large part by the individual's personal experiences and current concerns. The telling of stories in context allows the hearer's concerns in that context to influence his understanding of the story. For example, told in a completely different context, the story of the man who learned to catch flies with honey not vinegar could be interpreted as a story about how to trap animals successfully. In this case, the lesson of the story would be "If you want to catch animals, it is important to have the right sort of bait." However, in the context of a child behaving unpleasantly in order to get his way, the lesson of the story is allegorical. Honey represents pleasant behavior and vinegar represents unpleasant behavior. Thus, context can change the lesson of a story entirely. The appropriate context can help the hearer to understand the lesson that the teller intends.

In addition to influencing how the hearer understands the story, the context also influences how he remembers it. A hearer can store a story in his memory with respect to the context in which he heard that story. When he finds himself in a similar situation in the future, he can then recall the story and apply its lesson to the new situation. (This is an example of using stories to support case-based reasoning which I discuss in more detail later in this chapter.) So when people teach with stories, they take advantage of the role that context plays in helping a hearer to retrain the story.

The difference between telling stories to establish context and telling stories in response to context is the difference between the case method and case-based teaching. Case-based teaching is much less common in formal instruction because it requires a gifted teacher, one who can not only think of the right story or case to tell on the spot but who can present it effectively. Teaching with the case method is easier because cases can be selected and prepared in advance. However, as we shall see later in this chapter, the computer is especially well-suited to case-based teaching.

2.1.2 Learning from Stories: Case-Based Reasoning

When a person confronts a situation that he has never seen before, he will often refer back to some similar situation he has seen in the past, and decide how to behave based on the earlier experience. This process of learning from prior experiences is called *case-based reasoning*. The theory of case-based reasoning (Kolodner, Simpson, and Sycara-Cyranski 1985; Riesbeck and Schank 1989; Schank 1982) argues that many situations are too complex for people to deal with by reasoning from first principles. Instead, they reason using previously stored cases. The theory of case-based reasoning was proposed as an alternative to the style of reasoning employed by rule-based expert systems (e.g., DENDRAL (Lindsay et al. 1980), PUFF (Feigenbaum 1977), MYCIN (Shortliffe 1976)). In this earlier model of cognition, knowledge is expressed in the form of simple rules, such as the following from MYCIN, an expert system for medical diagnosis:

```
If (1) The infection which requires therapy is meningitis,
   (2) Organisms were not seen in the stain of the culture,
   (3) The type of infection is bacterial,
   (4) The patient does not have a head injury defect, and
   (5) The age of the patient is between 15 years and 55
       years
Then: The organisms that might be causing the infection are
   diplococcus-pneumoniae (.75) and neisseria-meningitidis
   (.74). (Clancy, Shortliffe, and Buchanan 1984)
```

In a rule-based understander, when its observations match the *if* clause of a rule, "the lefthand side," then the "righthand side," the *then* clause gets asserted. This new belief is added to the knowledge of the system in hopes of triggering another rule. The system reasons by building up long chains of beliefs constructed from the application of these rules.

However, even common, everyday problems can contain a multitude of factors with messy interrelationships. In many of the spheres in which people operate, there is no strong theory that neatly connects antecedents and consequents. Therefore, rules can not be constructed that will make the explanations and predictions necessary to understand the domain. For example, the social world that we inhabit is too complicated to reduce to rules, yet teenagers are able to predict reasonably accurately how their father will react to news of a smashed up car and adults can explain why their boss did not promote a colleague. In complex domains like these, reasoning from first principles in the style of a rule-driven expert system becomes either impossible or impractical. Instead, in these situations people reason from cases. They use similar situations from the past to deal with the current situation. Therefore, in a complicated decision such as whether or not to send U.S. troops to Kuwait, one naturally considers previous cases like the U.S. involvement in Vietnam and Panama. Researchers have uncovered extensive psychological evidence to demonstrate the pervasiveness of case-based reasoning. In domains as varied as car mechanics (Ross 1989), medicine, catering (Kolodner 1991), and architecture (Klein 1988), researchers have shown people solving problems and making decisions by reference to previous cases.

This process of case-based reasoning has three steps:

1) Extract key features of the current situation;
2) Use those features to retrieve similar cases from memory;
3) Use the similarities of and differences between the current situation and the retrieved case to determine an appropriate action in the current situation.

If people reasoned from first principles most of the time, then you would naturally teach them by presenting them with the sorts of rules they could use to construct large chains of connected inferences. However, since most subjects worth teaching are too complex for a rule-based approach and case-based reasoning is an important part of the way people deal with these subjects, it is important to teach them in a way that will assist the natural process of case-based reasoning. While the prevalence of the case method in professional education reflects this recognition, much of K-12 and college education does not. In fact, those subjects that are taught in a rule-driven fashion are the subjects in which students most rarely achieve any deep understanding. In most public schools, math and science are still taught as the memorization of formulas and algorithms, and our failure to compete with the rest of the world in these areas is now considered a national crisis.

Therefore, a central goal of teaching must be to support case-based reasoning by giving students cases they can reason with. Students can effectively gather cases through either first or secondhand experience. In schools, firsthand experience has traditionally been provided by laboratories, demonstrations, role-playing, simulations, and other hands-on activities. These experiences are important for helping students to build up personal case libraries. However, firsthand experience is not always feasible or even advisable, so cases can be provided secondhand through stories. Because stories resemble firsthand experience, they enable students to build personal cases libraries using the same natural learning process that they do when learning from experience.

While experience and stories are both effective for providing cases, combining firsthand experience with stories can improve on either experience or stories by themselves. An advantage of the case-based teaching architecture described in Chapter 3 is that it provides students with an opportunity to mix experience with stories. In a learning situation in which students receive both experience and stories, stories can provide them with cases that go beyond what they can experience firsthand. Stories can reinforce students' experiences, and they can help students to explain their experiences. In combination with experience or alone, stories can provide students with cases to reason with, without it being necessary for them to gain firsthand experience of everything they learn. However, effective teaching must be more than just giving students large numbers of cases. It must assist students to organize their memories appropriately.

Teaching Toward Case-Based Reasoning: Cases Plus Indices

In order to perform case-based reasoning effectively, a person must have a large library of cases that covers an adequate variety of situations. The accumulation of such a library is one of the goals of human learning in natural settings. For example, one of the things that people do when they start a new job is to observe their boss as much as possible. They do so in order to build a case library of situations in which their boss has acted, so that they will be able to predict in future situations how the boss will act. However, having a large library with wide coverage in and of itself is not sufficient. It is also necessary to have the case library organized, or indexed, so that appropriate cases can be retrieved at the moment when they are most useful. For example, an employee who organizes his observations of his boss according to the phase of the moon is less likely to have a usefully indexed case library of the boss's behavior than an employee who indexes his observations of the boss according to the latest sales reports. In a case-based memory, cases are organized according to features that describe situations in which the case may be useful. The collection of features that label a particular case are called an *index*. As the example above shows, a case library is only as useful as the indices that organize the cases. If cases are stored under irrelevant or incorrect features, then the reasoner will not be able to retrieve those cases when they might be useful in the future.

Therefore, an important objective of teaching with stories must be to help the student develop appropriate indices for the cases in his memory. When people learn from experience, they pay attention to the circumstances in which an experience occurs. This context helps an individual to store an experience as a case in his memory. The context provides the learner with features that he can use to index the case. When he finds himself in a situation that has features that match the context of the earlier experience, he is able to retrieve the earlier case to help him to decide how to act in the new situation. When teaching with stories, the context in which a story is told can help provide features to index with, in the same way that the context does when someone learns from experience. Effective teaching with stories takes advantage of context to help the student index cases effectively in his or her own memory. For example, if a student makes a mistake in solving a problem, that mistake establishes an important context for the student to hear a story that will help him to avoid that mistake in the future. The features that characterize his current situation, the problem he's working on, the mistake he made, form the basis for indices to that story. If the student indexes that story in his memory based on this context, then he will be able to retrieve that story when he finds himself in the same situation in the future. Having retrieved it, he will be able to avoid making the same mistake or to recover from that mistake if he repeats it. Thus, the context in which he hears the story provides features that help a student to index that story effectively. A well indexed story can be retrieved when it becomes useful for dealing with novel situations.

To summarize, one of the most compelling reasons to teach with stories is the role that these stories can play in supporting case-based reasoning. Stories are second only to firsthand experience in providing people with cases for reasoning with, especially when firsthand experience is impossible or impractical. Effective story-based teaching is more than just presenting an arbitrary collection of stories. The stories must cover a wide enough range of situations and they must be presented in a way that helps the student to index them appropriately for future use.

2.1.3 The Computer: A Platform for Teaching with Stories

While teaching with stories is natural and effective for human teachers, modern computer technology offers several qualities for teaching with stories that improve on the capabilities of any individual person. The first advantage of the computer for teaching with stories is its capacity for multimedia story presentation. A computer can employ graphics, animation, sound, and on-screen video, as well as ordinary text, for the vivid presentation of stories. More important, a computer has instant access to its library in a "random access" fashion. This means that a computer can locate and present a particular story at the instant that it determines the story is valuable for the student. Human teachers can draw on all of these media for story presentation, too, but they can not instantly retrieve the film, filmstrip, or graphic that they need. Typically, teachers need to prepare days or weeks ahead in order to have the equipment and material available to make a media presentation. Teachers cannot take advantage of unanticipated opportunities for teaching with these sorts of media. In the current environment, they can teach with audiovisual stories, but they cannot do the sort of instant-access, opportunistic teaching with stories case-based teaching requires and that is native to a multimedia computer system.

Along with a wide range of dramatic presentation media and instant access, computers have the advantage of large storage capacity. With mass media storage devices and networking, computers now have huge libraries of information available to them. Storage capacity that was either impossible or unaffordable a few years ago is commonplace today, and the trend gives every indication of continuing for the foreseeable future. The implication of this technology for teaching with stories is that a computer can have many more stories at its disposal than a teacher ever could. When a story gets indexed in a computer system that teaches with stories, the story is permanent, regardless of the other stories entered into the system. By necessity, humans do not operate that way. They forget. New knowledge crowds out old knowledge, stories get confused with each other, and little-used information eventually gets lost. Computers do not have the intelligence for dealing with knowledge that humans have, but on the other hand, their memories are insensitive to the size of their knowledge base. They do not forget old stories when new ones are added.

With the computer's unique ability to store large amounts of data comes an opportunity to increase the number of perspectives that can be taught. Humans are generally unable to maintain conflicting perspectives in their minds, but teaching systems can be built that incorporate the experiences and stories of many individuals regardless of conflicts in perspectives. Once they are stored in the computer they can be widely distributed at little additional cost. Often computers are advertised as being an improvement in education because a student using a computer can receive individualized instruction. This, advocates claim, is like a one-on-one situation and, therefore, better than the one-to-many teacher-to-student ratio that is the rule in modern education. However, when a system can include the stories of many individuals, it improves the teacher-student ratio to many-to-one, so a single student is now able to benefit rom the experiences and expertise of many teachers. Since the stories collected would be the best available, the student will have the benefit of a large number of the best possible teachers. Expertise and experience are generally scarce resources. A primary advantage of computer-based teaching with stories is that stories can be collected once and then widely distributed. This makes access to these sources of expertise and experience feasible and affordable for large numbers of people who wouldn't ordinarily have any access to them at all.

The computer is still a long way from approaching human intelligence. It will not be a replacement for human teachers for a long time. However, it does have certain capabilities that are ideally suited for teaching, and teaching with stories takes advantage of these capabilities in ways that other attempts at computer learning environments have failed. For example, researchers in artificial intelligence have worked on intelligent tutoring systems (ITS's) for over two decades (Sleeman and Brown 1982; Wenger 1987). The objectives of these systems are to instruct students in subject areas by maintaining a model of what the student ought to know and contrasting it with what it believes the student does know. This strategy is based on observations of human tutors. As a strategy for tutoring it makes sense, and it may be effective across a large range of subjects some day. However, it is restricted by the computer's knowledge, ability to interpret the student's actions and its ability to react flexibly. An ITS must really be an expert in the field that it is teaching, as well as an expert teacher. For most fields (with the possible exception of algebra and computer programming), this is beyond the current capabilities of artificial intelligence. In order to teach with stories, however, a computer program does not need to be nearly so intelligent. A great deal of the teaching and the expertise can be encoded in the stories, and the system need not be able to understand the stories themselves. Unlike an ITS, a system that teaches with stories does not need to be expert enough to understand everything it is teaching. It does not need to understand its stories; it only needs to know when to tell each story. In this way, the computer becomes less of a tutor and more of a librarian. By observing the student it gains an idea of what knowledge would be useful at the moment, and it presents the most appropriate story or stories from its library.

On the other end of the spectrum of computers in education is Computer-Assisted Instruction (CAI). These systems guide students through a series of exercises and instructional materials. Often practice is mixed with presentation of material, and the student progresses or receives remediation depending on how well he or she performs. These systems are effective and improve on workbooks or other more traditional materials to the extent that they respond dynamically to the student and provide immediate feedback. However, they miss the opportunity to take advantage of the computer's full range of capabilities. Because they typically present lessons in the form of rules and generalizations, they do not capitalize on the learning that can take place when a system has a large body of stories and cases to draw on for teaching. They take advantage of the computer's responsiveness, but the teaching they do tends toward teaching through abstractions not cases and they lack the large libraries necessary to support case-based reasoning effectively.

2.1.4 A Definition of *Story*

Because this research looks at stories in a new light, it is important to explain what constitutes a story for the purposes of this research. Stories are studied and discussed in many academic disciplines from literature to psychology to drama. In fact, the last few years have produced a resurgence of interest in stories in many forms. The ancient art of storytelling, practiced since before the days of written language, has grown in popularity lately, especially among those unhappy with the increasing isolation from personal contact and live performance in the modern world of TVs, radios, telephones, and facsimile machines. Professional storytellers are becoming increasingly popular, and storytelling festivals are held throughout the United States. In addition, in academic circles there has been an outpouring of works that use the story as an unit for gaining insight into human nature for various purposes. In the field of psychology, for example, Robert Coles (1989) has used the stories told by the mentally ill to help in psychotherapy. Vivian Gussin Paley (1990) has focused on the stories told and acted out by young children as an integral part of her teaching, arguing that they provide a window into the mental processes and social development of children. Thorndyke (1975) has studied human understanding of stories and the impact of story structure on comprehension. In *Tell Me A Story* (Schank 1990), the basis for the research described in this dissertation, Roger Schank presents a theory of human intelligence in which stories play a fundamental role in organizing human memory, and in determining human behavior. In the social sciences, researchers have studied stories and the role they play in forming, delineating, and preserving cultures. The art of telling stories is taught in schools of performance, and of course the analysis of stories lies at the heart of the study of literature and of writing. In the field of literature, volumes have been written on the nature of stories, how to write them, what constitutes an effective story, and how to read and analyze them according to various doctrines including Freudianism, Marxism, feminism, and deconstructionism.

Even within the field of education, stories have received a great deal of attention. Much has been written on the use of stories in the classroom, but the focus has primarily been on the traditional "story hour", a time that is set aside for the telling and discussion of a story. This form of teaching with stories is a longstanding practice and an effective one. Indeed, listening to a teacher read or tell a story in an elementary school classroom is one of the strongest and warmest memories of this period for many people. An extensive literature advises teachers on how to select, tell, and adapt stories to be told during the story hour (e.g., Bryant (1905), Fitz-Gerald (1971), De Wit (1979), Jones (1970)). However, as Paley (1990) argues,the division between "story hour" and the rest of the day is artificial. In her view, the school day consists of the successive telling, revising, and retelling of stories by both students and teachers. An examination of teaching practices reveals, teachers use stories as teaching aids not just during "story hour" but throughout their day, across all subject areas and all levels of instruction.

Since stories are not just told during story hour but are thoroughly integrated into ordinary teaching, the term *story* can no longer be defined in education as "what gets told during story hour." It must be defined in a way that enables one to distinguish teaching with stories from other forms of teaching. Through the following criteria, I develop a definition of a story and I show how the definition can be used to differentiate teaching with stories from other teaching.

Criterion 1: A story must have characters, real or fictitious.

As an example of how characters distinguish between stories and non-stories, consider a chemistry lesson on the nature of volatile substances. In both cases, the teacher presents an example of the same phenomenon to show the force of an explosion that can arise when some volatile gas is exposed to oxygen at room temperature. In the non-story example, the teacher says, "For example, if one liter of this gas were mixed with oxygen at room temperature, there would be an explosion with the force of four sticks of dynamite." In the story example, the teacher says, "For example, when I was in college, a careless graduate student allowed just a fraction of a liter to come into contact with the air in his laboratory, and there was an explosion that destroyed all the furniture in the room and blew out part of the nearest wall. I don't think the grad student survived." Even though both are examples that describe the same phenomenon, the story has a character in it, the graduate student, while the non-story example has no identifiable character. The distinction between stories and non-stories is important in cases like these because teaching with examples is not necessarily a form of teaching with stories. The presence of characters is important for learning from stories because identification with a character in a story enables the hearer to use the same natural process of learning that he would use when learning from firsthand experience. Identification with characters allows for egocentric understanding of stories. A character can be any creature, real or fictitious, that acts with intention. Thus, an extraterrestrial, an animated teapot, or a housecat can all be characters.

Criterion 2: A story must have an identifiable incident.

This may be the most obvious criterion for a story, but it cannot be overlooked. The characters in a story must be involved in some event. Thus, saying your uncle was the stingiest man alive is not telling a story, but describing an incident in which he drove sixty miles to save money on toilet paper is. Again, the importance of identifiable incidents is that it makes stories resemble actual experience. The similarity to experience enables someone to learn from stories in the same way that he or she would build a case library from firsthand experience.

Criterion 3: A story must deal in specifics. It is not a story if it is couched in abstractions.

This criterion may be subtle, but it is important because it distinguishes between generalizations or rules and stories. For example, in the domain of animal adaptation, a teacher could say, "Pikes have nostrils that sense pressure changes in the water and enable them to detect possible prey." This has characters, pikes, and it has an identifiable incident, sensing pressure changes to detect prey. However, because it describes pikes in general, not any specific pike, it is not a story; it is a generalization. The reason that the videos in Creanimate qualify as stories is that they depict specific animals in the midst of specific events. This requirement of specificity does not exclude stories that use specifics to represent abstractions. Stories that use symbolism and metaphor, such as fables and allegories, are both common and effective. In these stories, specific characters or events represent a class of individuals or events, and the storyteller expects the hearer to understand the allusion. So, a story must be expressed in terms of specific characters and events, but its intended meaning may be abstract. Aesop's fables, for example, are stories about animals, but their intended meaning is always to teach a general lesson about human nature and appropriate activity.

Criterion 4: A story must have a point.

It is possible to tell a narrative that describes specific individuals in the midst of specific incidents, but has no point. A story must contain some lesson for the hearer. The hearer must be able to glean some *useful* information from the story. The criterion of usefulness is important, for it distinguishes a non-story that contains information the hearer does not know but does not need to know from a story that contains information that the hearer does not know but has a need to know. For example, I could describe for you, all of the events that occurred to me between the time I awoke this morning and the time I left my apartment. This would meet criteria 1-3 for a story, it would even include new information for you, however, except in truly extraordinary circumstances, you would have no use for this information. To you, the story would have no point. The points of stories can vary widely, from the lessons on how to behave of fables to the specific principles that videos of animals illustrate.

Criterion 5: Stories can be told or observed but not experienced.

This criterion serves two important functions. It expands the traditional definition of stories beyond the spoken or written word, and it excludes any firsthand experience. First, a story can be told or observed. In other words, a story can be presented as a narrative in which one person conveys it to another through text or speech, it can be dramatized as in a play or animation, or it can be reproduced from reality as in a captioned photograph, a film, or a videotape. However, a story must be communicated not experienced. For example, if I walk down the street and get mugged, that is an experience. If I come home and describe it to my wife, that is a story. If a camera crew happens to be there and captures it on videotape, then they had an experience, but if the district attorney presents the videotape to a jury as evidence, then he is presenting a story.

In order to evaluate the effectiveness of teaching with stories accurately, it will be important to distinguish between teaching with stories and other forms of teaching. The criteria above serve to differentiate stories from experience on one end, and from other forms of communication on the other. Stories are more effective than other forms of communication to the extent that they enable the hearer to build a case library for case-based reasoning in the same way as he would from firsthand experience.

2.2 Teaching with Questions

While stories can play an important role in effective instruction, they are but one of the tools at a good teacher's disposal. Because case-based teaching uses stories to respond to context, establishing the appropriate context for storytelling is crucial. The Creanimate system uses questions to help establish this context. Therefore, in this section I provide a rationale for the use of questions in teaching.

The reasons for asking questions in the course of teaching fall into two categories. First, the ability to ask good questions is essential to learning and understanding. Teaching with questions is a way of modeling a behavior for a student to learn. Second, asking questions is a valuable device for increasing the effectiveness of a learning interaction. In the next two sections, I explore these reasons for using questions in teaching.

2.2.1 Questioning and Understanding

The ability to ask good questions is fundamental to the processes of learning and understanding. For example, Chi et al. (1989) have observed students in the course of teaching themselves from textbooks and has identified questioning as a skill that distinguishes between good and poor learners in such self-teaching situations. An important aim of the Creanimate system is to help students learn to ask good questions by modeling the behavior for them. Recent educational research (Collins, Brown, and Newman 1989) has advocated modeling as a way of helping students to develop desired behaviors. Modeling consists simply of performing a behavior for a student, so the student can internalize that behavior. This has been identified as an integral step in apprenticeship learning (Collins, Brown, and Newman 1989). Collins and Stevens (1982) and Brown and Palincsar (1989) have described the effective use of modeling to teach questioning in inquiry teaching and reciprocal teaching respectively. Admittedly, the cognitive apprenticeship approach advocated by Collins, Brown & Newman (1989) includes, in addition to modelling, coaching, scaffolding, articulation, reflection, and exploration. Therefore, one of the issues to be addressed in the empirical research on Creanimate is: Can modeling be effective to teach good questioning behaviors in the absence of the other elements of apprenticeship learning?

The primary reason to teach students to ask good questions is the role they play in understanding and expertise. An expert is someone with a good understanding of a topic. While we usually think of an expert as knowing the answers in his field, the ability to ask good questions is at least as important for expertise. Knowing the right questions to ask is an important element of expertise because it is critical for dealing with unfamiliar situations. What differentiates an expert from a novice in any field is knowing the questions to ask to begin to understand novel phenomena (Schank 1986). For example, if you present a geologist with a land formation that he has never seen before, he or she will know the right questions to ask to start constructing an explanation for how that land formation came about. Explanation, the central process of understanding, alternates between asking questions and constructing plausible answers. But, where do those questions come from? The questions that one asks in the process of explanation in any domain derive from knowledge of the basic relationships that underlie that domain. These questions, called *explanation questions* in the terminology of Schank (1986), connect the concepts in the domain together.

As an example, consider a naturalist exploring the Amazon jungle who comes across a lizard with bright red streaks on his back and stomach and suction-cups on his tail. Never having seen anything like it, he starts to ask himself some questions

"Why does the lizard have those red streaks on his back? What do they help it to do?" he asks. "Maybe they are for camouflage," he thinks, and he starts to look around for similar colors. "But, if they're for camouflage, why would they be on both his stomach and back?" Just over his head, he notices a tree with unusual vegetation. It has long, thin red leaves hanging down. The leaves are exactly the same color as the lizard's streaks. "That explains the suction-cups," he thinks, "They allow him to hold onto those smooth branches and hang down like those leaves. But why would he hang down like that?" Just then, a small insect flutters by and disappears in front of his eyes. When he moves closer to see what happened, he discovers a similar lizard hanging down where the naturalist had previously seen only red leaves. It was this lizard that had caused the insect to disappear; he had grabbed it out of the air with his tongue.

This example highlights the role of explanation questions in understanding. The naturalist asks two of them, "Why does the lizard have those red streaks on his back?" and, "Why would he hang down like that?" These are explanation questions because they get at the basic relationships that underlie animal adaptation, the relationships among physical features and their functions and the relationships among animal actions and their survival goals. Thus, explanation questions exist on two levels:

Explanation question instance. An explanation question instance is a specific question. An explanation question instance can be generated by applying an explanation question category to specific concepts. For example, the explanation question category "Why does an animal have a physical feature?" applied to a lizard with red streaks produces the instance, "Why does that lizard have those red streaks?"

Explanation question category. The question category describes the basic relationship that an explanation question refers to without mentioning any specific concepts. For example, an explanation question category might be, "Why does an animal have a physical feature?"

Explanation question categories are important because they coincide with the important relationships in a domain. Someone who understands the explanation questions for that domain will organize his knowledge around those relationships. For example, a naturalist's knowledge would likely be organized around the relationships between physical features and their uses and between animal actions and their goals. Therefore, teaching a student the explanation questions in a domain is the same thing as teaching him the important relationships that underlie that domain. Furthermore, once a student learns these questions, he can teach himself about new things by asking those questions and organizing his observations according to the relationships that the explanation questions express. Knowing the explanation questions for a subject is more important than possessing a large body of facts about that subject, because explanation questions provide the ability to extend one's knowledge at will.

Teaching the explanation questions underlying animal adaptation is the central pedagogical goal of Creanimate. While an individual can understand the relationships within this subject area at ever-increasing levels of sophistication, we have identified five initial explanation question categories to teach students. As we will see in later chapters, these explanation question categories shaped the knowledge base of the Creanimate system, as well as its reminding strategies and dialogue management techniques. These explanation question categories are:

Why feature? The *Why feature?* explanation question attempts to connect an animal's physical feature with some action that the feature supports. For example, "Why do cheetahs have long legs?"

How Action? The *How action?* explanation question explores the same relationship as *Why feature?* However, *How action?* starts with an action performed by an animal and looks for the features that are necessary to perform that action. For example, "What does an eagle have to help it to fly?"

Why action? The *Why action?* explanation question attempts to connect a action performed by an animal with some survival behavior. It assumes that the actions that animals perform enable them to survive in some way. For example, "Why do female alligators dig holes?"

How behavior? The *How behavior?* explanation question is the reverse of *Why action?* It starts with a survival behavior and seeks actions that enable an animal to achieve that behavior. For example, "How do chimpanzees get food?"

Why behavior? Behaviors can be performed for more than one reason. For instance, finding shelter can be in the service of both keeping warm and avoiding danger. Therefore, the final category of explanation question is *Why behavior?* For example, "Why do bears fight?"

These five explanation question categories correspond to the relationships among three different types of concepts: features, actions, and behaviors. These basic concepts and the relationships can provide a student with a solid groundwork for the understanding of animal adaptation. The goal of Creanimate is to ask questions in a way that will help students to develop an understanding of adaptation according to these relationships and to learn to ask the questions themselves as they continue to learn on their own.

2.2.2 Questioning in an Instructional Interaction

The second reason for asking questions in teaching is that it helps make the student an active participant in a learning interaction. Responding to a question provides a student with an opportunity to think independently and creatively, in contrast to situations in which the student passively absorbs information transmitted at him. A great deal of research has examined the use of questions in classroom settings and found surprising limitations in their effectiveness (e.g., Dillon (1985); Winne (1979)). Many of these limitations, however, can be directly attributed to the social context of a classroom. In a classroom, only one or a few students actually have the opportunity to express their answers to a question. The remaining passive observers are not likely to expend much effort on generating answers that will not be pursued. In addition, when a student does answer a question in a class, he or she does so in a social environment in which failure may lead to embarrassment or humiliation. The classroom environment often produces a fear of incorrect answers that discourages creative thinking and causes questioning to be more destructive than helpful. However, in an individualized interaction the student can actively respond to questions free from the intimidating social pressures of a classroom situation. Therefore, thought-provoking questions can prove particularly effective in a computer-based learning environment. Questions activate students in two important ways. They provide students with opportunities to form their own hypotheses, and they evoke curiosity. Both of these sources of activation are especially effective in combination with the use of stories.

A thought-provoking question gives a student the opportunity to construct his or her own hypotheses. Forming one's own hypotheses is important because of the investment and perspective that they give the learner. First, a hypothesis increases a student's interest in any subsequent information that relates to the hypothesis. Commitment to a hypothesis provides a student with the motivation to attend to and consider relevant information Second, a hypothesis gives the learner a frame of reference for evaluating additional information. Instead of considering new information from an uncommitted perspective, a hypothesis gives a learner the ability to evaluate information with respect to the ways it supports or conflicts with his hypothesis. Finally, a hypothesis provides a student with a framework for integrating new knowledge into his existing knowledge. To the extent that new information agrees with his hypothesis, the student can integrate the information with his hypothesis in his memory structures. To the extent that new information conflicts with the hypothesis, he can refine or modify his memory structures appropriately. Thus, regardless of the correctness of a student's initial answer to a question, it provides an important context for subsequent learning. An educational system that asks questions can encourage students to explore the hypotheses they construct and to learn from the successes or failures of their hypotheses. Exploring their own hypotheses provides students with an unequalled opportunity for learning from information they encounter. In case-based teaching systems, such as Creanimate, this information can be provided through stories.

In addition to providing opportunities to construct hypotheses, good questions evoke curiosity. Curiosity, in turn, is a dramatic motivator for learning. Therefore, a well-formed, well-timed question can motivate a student to be an active learner. Specifically, an effective question can evoke the sort of curiosity that establishes a context for storytelling. A student who sees a story in the context of an unanswered question that is meaningful to the student's concerns will become an active viewer of that story. He will focus on the story as an opportunity to satisfy his curiosity and he will view it as a source of an answer to his current question. This active viewing contrasts sharply with the way the same student might attend to the same story in the absence of a question. However, the effectiveness of questions to evoke curiosity is not limited to storytelling situations. A well-timed question can help a student to make new connections using his previous knowledge, or to conduct his own inquiry.

2.3 Summary

The use of stories to teach is common among both parents and formal educators. Teaching with stories generally falls into two categories, stories that are used to establish a context and stories that teach lessons within a context. The former is the technique of the case method commonly used in professional training, and the latter is the technique of case-based teaching.

Teaching with stories is effective because it supports the natural process of case-based reasoning. In order to perform case-based reasoning, it is important to have a large library of cases that cover an adequate variety of situations. One of the goals of natural human learning is to accumulate such a library. However, having a large library with wide coverage, in and of itself, is not sufficient. It is also necessary to have the case library organized, or indexed, so that appropriate cases can be retrieved at the moment when they will be useful.

Teaching with stories is ideally suited to the computer. Computers offer an unequalled opportunity to build large libraries of stories that can be accessed instantly. They can present stories in a wide variety of vivid fashions that take advantage of advances in graphics, animation, audio and video. Through their ability to store large libraries, they offer students exposure to the experience of many individuals, broadening their perspectives beyond what they can get in a normal classroom, a one-on-one teaching situation, or even a more traditional computer-based learning environment.

Good teachers also ask questions. Teaching with questions is important because questions can provide a student with the structure of a subject matter. In particular, explanation questions teach the basic relationships that are important for constructing explanations of phenomena in a domain. In addition, questioning is an important part of the self-teaching process. Students who learn to ask the right questions are able to learn from new experiences.

Questioning can also have an important impact on a learning interaction. Questions help to activate the learner to construct his own hypotheses and learn from them. In addition, thought-provoking questions raise curiosity which is a powerful motivator of learning.

Chapter 3

The Case-Based Teaching Architecture

Two basic pedagogical principles underlie case-based teaching. They are:

Active Learning. Effective learning takes place when a student is engaged in the active pursuit of tasks that provide him with motivation and opportunities for learning.

Learning from stories. When opportunities for learning arise, a student should be provided with stories that will help him to learn from his situation.

The case-based teaching architecture consists of two components, the *task environment* and the *storyteller*, each of which embodies one of these principles. In a case-based teaching system, a task environment engages the student in active learning, while a storyteller monitors the student's actions looking for opportunities to present stories. The stories that the storyteller presents help the student learn from the situations he encounters. In combination, the task environment and storyteller provide forms of knowledge that complement each other. The task environment helps a student to understand the structure of the domain under study, while the storyteller provides cases that support that structure. Taken together, the framework that the student learns from the task environment and the cases that he gains from the storyteller are mutually reinforcing.

In this chapter I present principles for the design of effective task environments and storytellers. In Section 3.1, I discuss the design of task environments that support active learning, and in Section 3.2, I discuss the design of storytellers that support learning from stories. In these sections, I present specific issues for the construction of effective case-based teaching systems. The discussions of these issues are accompanied by descriptions of how they have been resolved in Creanimate. In the final section, I bring the two components back together with a discussion of the learning that results when an effective task environment is combined with an effective storyteller.

3.1 Active Learning: The Task Environment

The principle of active learning is supported by theories of constructivist learning which argue that a student learns best when he is constructing understanding as a result of activity.[1] In active learning, a student generates his own hypotheses, explores them, evaluates and revises them as a result of self-generated actions. Active learning contrasts sharply with the kind of passive learning that we see in most traditional educational settings. In a lecture setting, students are not given the opportunity to generate their own hypotheses; they are presented with a description of someone else's hypotheses and how they were evaluated. Worse, students often are not exposed to hypotheses at all but only to conclusions. In this situation, students do not learn about the processes that lead to understanding. They learn facts dissociated from the processes that uncovered them. While drills and exercises make a student active, they rarely provide the valuable opportunities for learning that characterize effective active learning. In traditional exercises and drills, students lack the ability to control their own learning or to learn from timely feedback.

Active learning demands that the case-based teaching architecture provide an environment in which students can formulate, explore, and revise hypotheses at their own initiative. In the course of exploring these hypotheses, students must encounter valuable opportunities for learning. In addition, a task environment must be authentic to both the concerns and interests of the student and the real world application of the material being taught. To effectively engage a student in active learning, a task environment must also provide a student with control of his interaction, and with sufficient motivation.

3.1.1 Forming and Exploring Hypotheses

Historically, educators have used manipulatives, laboratories, thought experiments, essays, and open-ended discussions in order to provide students with settings in which they can perform active learning. All of these devices enable students to actively explore ideas. They provide a student with the means to express hypotheses, investigate their implications, and revise their hypotheses. An effective task environment provides these same capabilities.

In order to explore hypotheses, a student must be able to obtain feedback about them. An effective task environment provides a student with feedback that will enable him to observe the strengths and weaknesses of his hypotheses. The computer presents a unique opportunity for providing students with feedback in two respects. First, the computer can provide feedback in a wide variety of meaningful forms. Computers are uniquely suited to allow a student to manipulate objects and observe consequences. Valuable tools such as microworld simulations and visualization environments have been developed to take advantage of the computer's ability to provide rapid, appropriately formatted feedback. Computers can draw on a large variety of media for presenting meaningful feedback including audio, graphics, animations, and video. Second, the computer gives the software designer the ability to control the feedback that a student receives. With physical manipulatives and labs, feedback to the student is determined by the laws of nature. This limits the range of hypotheses a student can explore. In a laboratory, for instance, students cannot view the effects of microgravity. In addition, the laws of nature can make observation of interesting phenomena difficult. For example, chemistry experiments often go awry for reasons that are beyond the control of the

---

[1] While he apparently never used the expression *constructivism* , Piaget is generally credited with developing the theory of constructivist learning. (See, for example, Piaget 1954) For a discussion of constructivism and computer environments, see Forman and Pufall (1988).

student. When they do, students receive feedback that they cannot understand or use. Likewise, some natural processes, geological transformations, for example, proceed too slowly to even be observed naturally. However, a computer can provide an environment in which the results are tightly controlled, or in which the passage of time is accelerated or decelerated to match the processes under study to the attention span of the student. Not only can a computer environment make feedback easier to observe than in the natural world, it can generate outcomes that are incompatible with the natural world. This enables the student to explore all manner of hypothetical situations that will help him to understand naturally occurring situations better. Finally, the computer's feedback can be varied to suit pedagogical aims. The effects that a student observes as a result of his actions can be controlled in order to provide him with specific lessons.

Thus, the capabilities of a computer enable the task environment of a case-based teaching system to improve on traditional settings for active learning. An effective task environment gives the student tools to manipulate objects and observe results. It uses the computer's abilities to present information in a variety of formats in order to provide the student with feedback about the effects of his actions. It includes pedagogically sound techniques for controlling the feedback that the student receives in order to present the student with appropriate opportunities for learning.

Forming and Exploring Hypotheses in Creanimate

The Creanimate task environment gives students opportunities to form and explore hypotheses in two ways. The primary hypothesis, of course, is embodied in the animal that the student creates. This animal becomes the subject for an inquiry in which the student considers different aspects of its survival in the wild. In the course of a dialogue, a student examines his hypothesis through the consideration of questions about his animal like the one in the following excerpt from a transcript. The student has just decided to give his frog wings to help it fly, and the computer responds:

```
Having wings is not enough for your frog to keep itself up
in the air.  It needs more.
How else do you want to change your frog so it can keep
itself up in the air?
```
(transcript no. m1-4-29-15.41)

The thought-provoking questions that the program poses provide important feedback to the student. They ask students to consider the survival issues raised by their animals.

A student's formation and exploration of hypotheses continue as he proposes answers to the questions raised by the computer or considers answers suggested by the computer. When a student proposes an answer to an explanation question, he formulates a second hypothesis. For example, in response to the question above students have responded with both "make it smaller" and "make it larger." Creanimate provides positive or negative feedback to students' answers by either confirming that the answer is consistent with its knowledge or by telling the student that it does not know of any examples that support his answer. Both reactions provide opportunities for the storyteller to present stories.

In these ways, the Creanimate dialogue manager gives students the opportunity to generate creative hypotheses, and receive valuable feedback on those hypotheses. Students receive this feedback in two ways. They receive it through questions that ask them to explore their hypotheses further and through evaluations of their answers to these questions.

3.1.2 Opportunities for Learning

Before we can talk about establishing opportunities for students to learn, we must understand what triggers learning. People understand the world around them by maintaining a set of expectations and comparing these expectations with their observations (Schank 1982; Schank and Abelson 1977). At any moment in time, a person has a set of active expectations that are appropriate to the situation he is in. These expectations vary from the general, "I expect rooms to have doors," to the specific, "Rebecca orders chocolate and peanut butter ice cream at Baskin & Robbins." Through accumulated experience, people develop sets of expectations that are appropriate for particular situations. Schank (1982) divides these expectations into classes he calls scripts, scenes, and MOPs. When an individual enters a new situation, he looks for cues that tell him what kind of situation it is. When his expectations are confirmed by his observations, he moves smoothly though the world. When his expectations are violated, he becomes surprised, confused, or curious, all of which motivate him to learn. The result of such learning is a new or better refined set of expectations. Thus, people are prompted to learn when an expectation that they have is violated in some way. These violations are called *expectation failures* and they trigger *failure-driven learning*[2] (Schank 1982). Opportunities for learning occur at the moment that students experience expectation failures.

To promote active learning, a task environment should help students to develop increasingly sophisticated expectations, test them out, experience expectation failures, and revise their expectations accordingly. In other words, an effective task environment is one that provides students with opportunities to experience expectation failures. An implication of this theory of learning from expectation-failures is that students must feel comfortable making mistakes and must have opportunities to learn from their mistakes. A mistake is the manifestation of an expectation-failure and should be positively valued as an opportunity for learning. The task environment of a case-based teaching system should be sufficiently challenging and broad to enable the student to test his hypotheses and experience expectation failures. Once the student has experienced an expectation failure, then the storyteller can capitalize on this opportunity for learning by presenting a story that will help the student to learn from this failure. An advantage of computer-based learning environments is that students can feel free to make mistakes without the embarrassment or even humiliation that attends mistakes in a classroom. It is the rare teacher who is skilled enough to build a classroom culture in which students recognize the value of errors for learning and treat them as opportunities not embarrassments. However, a well-designed task environment can encourage students to take risks and be creative, observe the effects, and learn from their experiences. Students interacting with a computer can feel comfortable experimenting, free from the social pressures of a traditional classroom setting.

Opportunities for Learning in Creanimate

Opportunities for learning arise when a student has a misconception or is lacking some necessary knowledge. The result is an expectation failure. The Creanimate task environment gives students opportunities to form and express expectations in the form of both their invented animals and their answers to the program's explanation questions. They can then view stories that either confirm or violate their expectations. These expectations represent opportunities

---

[2]It is important to emphasize that the word *failure* in failure-driven learning refers to the failure of one's expectations not one's failure to achieve a goal.

for learning that the storyteller capitalizes upon. In the following example, a girl asks for a frog that can dance. The dialogue manager asks her,

> We need a reason for your frog to dance. Is there a reason you want your frog to dance?

The student chooses to ask for suggestions.

> **Student: Give me some suggestions.**
> Bees dance. Do you know why bees dance?
> (I think this video is amazing.)
> **Student: Show me a video**
> *[VIDEO: "BEE DANCES"]*
> Now, do you know why bees dance?
> **Student: Show me the video again**
> *[VIDEO: "BEE DANCES"]* (Transcript no. 5-8-13.59)

It is reasonable, without reading too much into this transcript, to conclude that the questions in this transcript exposed gaps in the student's understanding. Because the student asked for suggestions, we can surmise that she does not know, or is at least uncertain of, reasons that animals dance. The question exposed a gap in her understanding and established a context in which she would be motivated to learn reasons that animals dance. This opportunity for learning is exploited by the storyteller which prepares to present a story that shows why bees dance. This reason is presented in the form of a suggestion of why her frog might dance. In the course of introducing this story, the storyteller may have induced another expectation failure, this one more specific. It asked the student if she knew why bees dance. It is likely that she did not even know that bees dance, much less why they dance. This combination of expectation failures serves to raise the student's level of interest in the story. In fact, the complete transcript shows that she chose to view the video twice.

Opportunities for learning in Creanimate multiply as a student resolves the issues raised by the dialogue manager. Whenever a student settles an issue by answering an explanation question, he effectively refines the design for his animal. Each refinement in turn raises new issues for the well-adaptedness of the animal, presenting new opportunities for learning. The pursuit of these new issues will lead to further refinements, further new issues, and, of course, more opportunities to learn from stories.

3.1.3 Authenticity

Authenticity is increasingly being recognized as an important factor in an effective educational interaction (Brown, Collins, and Duguid 1989; Collins, Brown, and Newman 1989; Leinhardt 1987; Schoenfeld 1992). There are two issues for authenticity in a task environment: authenticity of the task to the student and authenticity of the task to the world. Authenticity to the student requires that the task relate to meaningful goals and interests of the student. Meaningful goals are vital because they allow the natural inclination that a student has to achieve some goal to carry over into motivation to accomplish the instructional task. As a result, authenticity leads to better understanding and retention of the material encountered. If the task is one that the student has both authentic interest and goals in, then the student will attend to the task more. In addition, a student's interest indicates that he has a better developed set of knowledge structures to which he can connect new knowledge. His goals mean that he will work harder to make these connections. Thus, to provide authenticity to the student, a task environment should connect with the natural goals and interests the student brings to the interaction.

The other side of authenticity is authenticity to the world. Authenticity to the world occurs when knowledge or skills are taught in a context that is consistent with the way they will be encountered and employed in the real world. Research has exposed the difficulty that people have applying the knowledge learned in one context in a different context. For example, studies have shown students' inability to transfer the physics they learn in schools to natural settings in which the knowledge applies (Halloun and Hestenes 1985; McCloskey, Caramazza, and Green 1980). This research is evidence for what Whitehead (1929) called *inert* knowledge. Inert knowledge results from artificial learning situations, because these settings do not help learners to understand the usefulness of their knowledge in natural settings. As a result, they are unable to recognize opportunities to apply their knowledge when it would be useful. The solution proposed by many current researchers is authentic learning situations, situations that are true to the settings in which practitioners actually employ the knowledge (Brown, Collins, and Duguid 1989; Collins, Brown, and Newman 1989; Leinhardt 1987; Schoenfeld 1992). This contrasts with the way subjects like mathematics are typically taught in our schools, in which basic skills are taught completely divorced from any context that indicates how the skills might be used. The result is that students learn the skills, but because they don't know how or when to apply them, they fail to use them in situations when they should.

One particularly effective example of an authentic learning activity is the Jasper project (Cognition and Technology Group at Vanderbilt 1990) which happens to also incorporate the use of stories. The Jasper project presents complex problems in mathematics to elementary and middle school students through an extended video story. At the conclusion of the video, the protagonist, an adolescent himself, is faced with a problem that can only be solved through a multi-step mathematical solution. The students work through the problem step by step, developing and using skills in context. The video story provides a context that is both authentic to the students because they identify with the protagonist, his goals, and his problem and is authentic to the world because the mathematical skills and knowledge are employed in a context that is faithful to the way those abilities would be called for in the real world.

To summarize, effective task environments must be authentic to both the student and the world. Authenticity to the student requires that a task connects with a student's own goals, interests, and concerns. Authenticity to the world requires that learning takes place in a context that is faithful to the way in which the knowledge will be encountered and used in the real world.

Authenticity in Creanimate

The task of creating a new animal is a natural one for children. As a natural part of their play, children invent new worlds and populate them with creatures of their own invention. Often when left to their own devices, children will draw pictures of their invented animals. Therefore, Creanimate is an authentic activity for them. The authenticity of Creanimate to the world is not as readily apparent, however. In fact, the Creanimate task environment is modeled very closely on the authentic practice of science. The practice of science consists of proposing creative hypotheses and exploring them. As we've already seen in this chapter, Creanimate provides the student with that opportunity. In addition, the practice of science in general and the study of animals in particular center on the activity of questioning. When presented with a novel phenomena, scientists rely on the explanation questions that define their discipline to construct plausible explanations. Creanimate leads students through the same process, using the same types of explanation questions that a naturalist would ask if he discovered a new animal somewhere in the wild.

Finally, the design of a new animal adheres to a basic process of experimental science. When a metallurgist wants to understand the properties of a material, he perturbs it from its natural state in various ways. He bends it to measure its rigidity, he stretches it to measure its ductility, he applies a current to measure its conductivity. In an analogous process, Creanimate gives children the opportunity to learn about animals by perturbing them from their natural state. Students learn about an animal by changing it from its natural state, and they learn about the features and activities of animals by giving them to animals that do not normally possess them. In response to their creating, the Creanimate dialogue manager leads students through an inquiry process that is faithful to the authentic practice of science.

In evaluating the authenticity of Creanimate, it is important to look past the fantastic nature of a task in which a child creates his or her own animal to see the scientific structure that underlies the interaction. Creanimate uses the natural interests and inclinations of children to draw them into an activity that allows them to practice scientific inquiry while they have fun. In the process, students engage and take pleasure in the creative thought that accompanies all real science.

3.1.4 Student Control

Active learning demands that a student be able to form, evaluate, and revise his own hypotheses. In other words, the student must control his own learning. This belief is the keystone of the discovery learning paradigm (Bassett 1970; Foster 1972; Shulman and Keisler 1966). The goal of discovery learning is for the student to direct his learning guided by his own natural interests and motivations. However, in practice discovery learning has two problems. First, with no knowledge of the subject to be "discovered" a student does not know which paths to pursue and which to ignore. Second, with little knowledge of the subject and no personal goals with respect to the subject, a student has no motivation to explore. Thus, the two challenges of active learning with respect to the student are:

- How to give the student control without dooming him to explore aimlessly.
- How to motivate the student to be an active learner.

The first challenge of active learning is a true dilemma. We want the student to control his own learning, but we don't want him to wander in the aimless style of unguided discovery learning. The solution lies in a cleverly engineered task environment that allows the student to control his interactions within the environment yet ensures that he will encounter appropriate opportunities for learning. One way to ensure this is to build the lessons of the system into every possible path that a student might choose. Thus, the student is given the opportunity to form his own hypotheses and control his interaction by choosing the actions he wants to take, but regardless of the hypotheses he forms and the choices that he makes, he should encounter situations that expose him to the important lessons the system is designed to teach. In Creanimate, the primary educational goal is to teach the student the important relationships that describe the makeup of all animals, i.e., physical features support actions and those actions contribute to survival behaviors. The student is given extensive control over his interaction, and yet, whichever animal he chooses to construct, or however he answers the questions posed by the system, he will be exposed to these same relationships.

A second approach to student control is to give the student a sufficient degree of control without giving him the freedom to wander aimlessly. For example, a student may be given a range of options at any point that is broad enough to impart a strong sense of freedom.

However those options may be selected to ensure that the student will still experience appropriate opportunities for learning. In this approach, the module that manages the task environment carefully selects options for the student based on pedagogical goals. In such a task environment, a student can exert a great deal of control over his interaction without being aware that the task environment is constructed to prevent his wandering in the aimless style associated with attempts at discovery learning.

Student Control in Creanimate

Creanimate gives the student an enormous amount of control of his or her learning interaction. Student control was a central concern in the design of the task environment. As a result, the initiative lies entirely with the student. The student determines the initial direction of the dialogue by choosing an animal to create. The ensuing dialogue responds directly to that choice and any subsequent choices made by the student. In the course of each dialogue, the student is encouraged to test his or her own ideas, however, students are also free to seek suggestions from the dialogue manager. In the course of any dialogue, several answers to an explanation question may be considered. After discussing each answer, the student has the opportunity to commit to any of the answers he has seen so far or he may choose to see more. Through the decisions the student makes in shaping his animal, he controls his own learning interaction.

In addition, the user-interface of the task environment is designed to give the student additional ways to exert control. The bottom of the Creanimate screen is devoted to a set of six "User Control Buttons." These buttons are shown in figure 24.

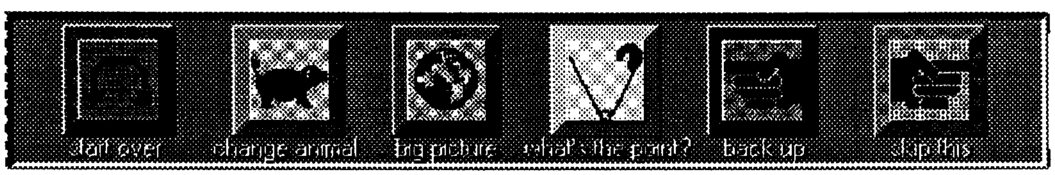

Figure 24. The button palette from the Creanimate user interface showing the User Control Buttons. The buttons are labeled, from left to right, *start over, change animal, big picture, what's the point?, back up,* and *skip this.*

The User Control Buttons in Creanimate are drawn from a set of sixteen buttons proposed by Jona et al. (1991). These buttons provide a student with an ability to express himself and control his learning interaction. They allow a student to request more information (*big picture, what's the point?*), change his mind (*back up*), jump ahead, (*skip this*) and change the topic (*change animal, start over*). Through these buttons, the student is able to convey information about his feelings that a human teacher can often read directly from a student's face or actions (and that intelligent tutoring systems devote enormous resources to inferring). In addition, they encourage the student to become more aware of his own learning processes and to act on this awareness. For more detail on the User Control Buttons in Creanimate and what they do, see Appendix D.

3.1.5 Student Motivation

The second challenge in encouraging a student to be an active learner is motivation. Unfortunately, motivation is the factor that is most commonly overlooked by artificial intelligence researchers in their attempts to build effective educational systems. A great deal of research has been conducted on student motivation (e.g., Deci 1975; Lepper 1978; Malone 1981; Keller 1983). This research indicates that for most purposes, motivation ought to be intrinsic. That is, the reward for accomplishing a task should derive from the nature of the task and the authentic results of executing that task, rather than from some external source. When the motivation is extrinsic, e.g., grades or compensation, Lepper et al. (1973) have found that it can reduce an individual's natural inclination to perform the task. In one study, they found that drawing, a task that pre-school children performed willingly in the absence of any reward, became unattractive after an intervention in which they were given an extrinsic reward for drawing. There is even evidence that suggests that the effects of extrinsic motivations are cumulative. Harter (1981) found that as students progressed from third through ninth grade in our current school system they reported behaviors that indicate progressively decreasing response to intrinsic motivation and increasing response to extrinsic rewards.

Fortunately, computer environments offer a new opportunity to provide intrinsically motivating, authentic tasks. Drawing from two strands of research in motivation, we have identified six critical factors for the design of a motivating task environment. The first four factors are taken from research reported in Malone (1981; 1980), Malone and Lepper (1987) and Lepper and Hodell (1989).

Challenge. A task environment should challenge students to work at the limits of their abilities. A challenging task is difficult enough that its accomplishment gives the student a sense of satisfaction but is not so difficult that it frustrates him or her. Ideally, the program that manages a task environment is able to vary the difficulty of the tasks in response to both differences in abilities between students and changes in the competence and confidence of an individual student.

Curiosity. According to Lepper and Hodell (1989) curiosity is elicited "by activities that provide students with information or ideas that are surprising, incongruous, or discrepant from their existing beliefs and ideas. "(91) Their characterization of curiosity agrees with the earlier discussion of the role of expectation-failures in triggering opportunities for learning. Once elicited, curiosity is a powerful intrinsic motivator of learning.

Control. Student control has already been described as being important for the actual learning process. However, control is also important for motivating a learner. As Deci (1975) argues, control increases a student's sense of self-determination which is intrinsically motivating.

Fantasy. The role of fantasy in motivation was first described by Malone (1981) as a result of his investigations of computer games and what makes them motivating. Fantasy, in and of itself however, is not sufficient for motivation. Malone distinguishes between extrinsic and intrinsic fantasy. A game in which a student moves a car around a track by answering arithmetic questions contains extrinsic fantasy because the skill of answering arithmetic questions has nothing to do with moving a car. On the other hand, a game in which students explore a cave by typing in instructions is an intrinsic fantasy for the skill of writing. Lepper and Hodell (1989) claim support for the motivation of fantasy from psychodynamic literature that discusses the emotional needs that fantasy fulfill and from Piaget who discussed the way fantasy adds cognitive structure to less engaging tasks.

In addition to the four factors described above, we have also adopted two from the work of Keller (1983) for the construction of motivating task environments.

Satisfaction. A task environment should provide results that are consistent with the student's effort, degree of success, and expectations. The combination of extrinsic and intrinsic rewards for performing a task must satisfy the student and live up to his or her expectations.

Relevance. Like curiosity, relevance is a characteristic of the task environment that is important both for the actual learning process and for motivation. Relevance is authenticity to a student's goals and interests. It requires that an active learning task connect to the goals and interests of the student. When a task is relevant to a student's personal goals and interests, these goals and interests will motivate the learning process.

These six factors provide guidelines for constructing a task environment that will provide a student with the motivation to be an active learner.

Motivation in Creanimate

The Creanimate task environment was designed from the ground up with the issues of motivation in mind. It provides challenge, curiosity, control, fantasy, relevance, and satisfaction in the following ways.

Challenge. The questions that the program poses were carefully examined to ensure an appropriate level of challenge for the target population. As part of this evaluation, students were observed using the program and interviewed to see how challenging they found the questions at several stages in the program's development. The questions were selected to raise issues that were appropriate for students in the target age group to learn. While some students were familiar with the answers to specific questions that the program posed, they were, by and large, still in the process of learning the importance of the underlying basic relationships.

Fantasy. The "create your own animal" task lends an ideal sense of fantasy to Creanimate. Unlike the overwhelming majority of science instruction in school settings, it encourages students to deal in a world of "What if..." Creanimate invites them to indulge in a fantasy, and it uses that fantasy as a point of departure for teaching important relations. The success of commercial games like Designasaurus (Patterson 1990) is testimony to the natural inclination of children toward this task.

Relevance. Relevance requires that the task connect to the interests and goals of the student. A great motivational force in Creanimate's favor is the natural affinity that children have for animals. Of the 21 fourth graders we surveyed, 100% said they "like to learn about animals" and 81% said they visit zoos often. In addition, 57% said that they like to watch nature shows on TV. Clearly these students already possess the interest in and goal of learning about animals. The interests of these particular students were typical of the other age groups we have worked with.

In addition, Creanimate uses the student's investment in his animal to bring more meaningfulness to the task. The advantage for a learning environment of a task in which a student creates something is that the act of creation generates a personal investment in the invention. Once a student has been given an opportunity to create his own design, his ownership of the creation becomes a powerful motivator for the student to learn about the issues that effect his creation.

Curiosity. The student's investment in his animal helps to motivate a student not just through meaningfulness, but also through curiosity. The student's interest in the adaptability of his animal carries over into curiosity about the factors that determine its adaptability. This curiosity, in turn, leads to interest in the survival of similar animals and the basic underlying principles that determine adaptability. As I described in Chapter 2, the thought-provoking questions posed by the dialogue manager evoke and focus this curiosity. In addition, the stories themselves elicit curiosity. While the videos in Creanimate provide a great deal of information, they have an important secondary effect of raising new unanswered questions. These questions that children generate themselves serve as powerful motivators for learning.

Control. As I described earlier in the chapter, Creanimate gives the student the ability to control his own learning interaction in several different ways. In addition to enhancing learning, the empowerment that comes from a sense of control increases a student's motivation to pursue the learning task.

Satisfaction. The Creanimate task environment offers two sources of satisfaction to help maintain student's motivation. The first source is provided by the physical transformation of the student's animal. When the student selects an animal to change, he gets to see a picture of the animal as it looks in nature. As he resolves questions that modify the physical structure of his animal, he gets to see a picture of his new, modified animal.[3] The second source of satisfaction in Creanimate is the stories themselves. Videos of animals in the wild are vivid and absorbing. Children frequently reacted openly and loudly to the videos. Their reactions range from laughing at surprises, to recoiling from fright, to squeamish squeals, to cheering over battling rivals. Frequently, students use the "replay" button to watch the same video two or three times. While the concrete satisfaction of seeing their own animal transformed is an important part of the interaction, we have found that a student's focus usually shifts after a short period of time such that they begin to see the videos as the real reward of using the program. The satisfaction of "discovering" a good story motivates students to explore creative ideas that will enable them to see interesting and unusual stories.

3.1.6 Some Designs for Effective Task Environments

After considering the features that make a task environment effective, it might be helpful to examine some designs for effective task environments. As I stated above, an effective task environment should provide a student with an engaging task that offers the opportunity to form and explore hypotheses as well as experience expectation failures. Some architectures for task environments that offer these capabilities include:

- simulations
- problem-solving environments
- design environments
- diagnosis environment

Led by the flight simulator, simulation environments have become extremely popular. Originally conceived to provide training for situations that are too costly or hazardous to allow a student to learn through actual experience, simulations have spread to every sort of situation imaginable. While simulations make ideal task environments for case-based teaching, problem-solving, design, and diagnosis environments share many of the same advantages. Problem-solving, design, and diagnosis environments all provide the user with a set of computer-based tools and an environment for accomplishing the particular type of task associated with the environment. For example, a problem-solving environment to teach finance might allow the student to create balance sheets for hypothetical companies. The tools in this environment would include a spreadsheet and a database for managing financial data. A design environment might allow a student to design bridges or buildings. This environment would include appropriate computer-aided design (CAD) tools. A diagnosis system might allow a student to troubleshoot a power generation facility by providing him with data analysis software and diagnostic routines.

---

[3]The ability to show the student an image of his animal is an important area for future improvement. The current implementation uses pre-drawn artist's renderings of the modified animals. Plans are underway to give future versions of Creanimate the ability to generate images dynamically in response to students' actions or to provide students with a graphics tools that will allow them to create their own images of their animals.

Simulations, problem-solving, design, and diagnosis environments all make effective task environments because they provide the means to express and explore hypotheses and the feedback necessary to evaluate and revise hypotheses. Furthermore, they provide the student with goals within the environment that enhance his learning. These goals serve as motivators, and they provide a meaningful context for learning. All of these environments put the student in control because they respond to the student's initiative. Finally, each of these types of environments is authentic both to the student and to the world. They allow students to learn a skill or accumulate knowledge through the natural application of that skill or knowledge, and they provide environments for learning that are true to the way that the material used in the real world.

3.2 Learning from Stories: The Storyteller

In the case-based teaching architecture, the storyteller provides learning from stories. The artful combination of task environment and storyteller enables a student to benefit from stories in context. The particular strength of the case-based teaching architecture is the way the task environment provides context for learning from stories. As I discussed in the previous chapter, the context in which a story is presented has an important impact on learning. Context affects both the way the hearer understands a story and the way he retains it. In a case-based teaching system, the task environment supplies a meaningful context for hearing stories. This context helps the student to index that story in his memory as a case for future use.

In addition, a storyteller presents stories at the moment a student can use them. In this respect, a storyteller lives up to Papert's (1980) *power principle.* The power principle states that the material being taught "...must empower the learner to perform personally meaningful projects that could not be done without it." (Papert 1980, 54) Told at any other time, a student may not understand the value of the information in a story and he will not be motivated to retain it. However, when you tell a student a story at a moment when he can use it to accomplish a meaningful task, he will be motivated to learn from it, the context will help him to understand it, and the experience of using the information will help him to retain it better. How to construct a storyteller that is able to present stories in this way is the subject of this section.

As I argued in Chapter 2, teaching with stories supports case-based reasoning. In order to teach with stories effectively, a storyteller must be able to respond to opportunities for learning with stories that provide useful cases. Toward that end, a storyteller in a case-based teaching system must have the following three attributes: 1) stories that cover a broad range of educationally significant situations, 2) an organization for those stories to enable the storyteller to recognize when a story is appropriate for the student's situation, and 3) a story presentation style to help the student to index those stories properly in his own memory. These requirements raise the following major issues for the construction of a storyteller:

- Story selection.
- Communication with the task environment.
- Indexing vocabulary.
- Reminding strategies.
- Presentation of stories.

Before a storyteller can teach, it must have a library of stories. The *selection of stories* for an effective storyteller is a complex task. In order to recognize opportunities for learning and intervene with stories, the storyteller must be able to *communicate* with the task environment. To identify and retrieve appropriate stories requires a library of stories with an appropriate organizational scheme. This in turn requires an appropriate *indexing vocabulary.*

However, a well-organized library of stories is not enough; the storyteller must have strategies for finding stories when they are relevant. It must be able to translate a description of a situation in the task environment into a retrieval cue for a story from its library. This translation and retrieval process is called a *reminding strategy.* Finally, the storyteller must present a story in a way that helps the student to understand how the story is relevant to him and assist him in making the connection between the story and the context in which it is presented. This "bridging" process helps the student to index the story as a case in his own memory. In the remainder of this section, I discuss each of these issues for the construction of an effective storyteller.

3.2.1 Story Selection

The first issue for a storyteller is story selection—how to collect a corpus of stories that teach effectively and cover a sufficient range of storytelling situations. The factors that influence the story selection process come from both the nature of the the task environment and the availability of useful stories. The objective of story selection is to provide stories that cover as many opportunities for learning that arise in the task environment as possible. Therefore, the primary concern of story selection is coverage.

Coverage

When selecting stories, it is necessary to take into account the frequency of occurrence and importance of a particular learning opportunity. Learning opportunities that arise most often and that are crucial for subsequent learning have a more urgent need for stories than opportunities that arise less often or are peripheral to the educational priorities of the system. The reason that learning opportunities must be ranked is that there are always resource limitations restricting the number of stories that can be included in a particular case-based teaching system. These limitations can include the capacity of the storage media allocated for stories, the cost of acquiring, producing, indexing, and integrating stories, time constraints on system development, and limited availability of stories. Because they can vary enormously, cost and availability place an important constraint on the selection of stories.

The range of costs and availability for stories can be demonstrated by considering some systems developed at the Institute for the Learning Sciences that teach with stories. Some systems use first person narratives told by experts (e.g., GuSS (Burke and Kass 1992; Kass et al. 1992), TaxOps (Schank 1991), Trans-ASK (Bareiss and Osgood 1993)). Producing these systems typically requires that a team interview an expert record his or stories. Depending on the availability of the expert and his receptiveness to interviewing, this can be an expensive and difficult process. Other systems use stories that can be drawn from already existing archives. The footage used in the Creanimate system is taken from such film libraries. Advise the President, an ILS system that teaches political science, uses footage from old television news shows and presidential press conferences. While hundreds of thousands of hours of footage lie untapped in archives, they are often poorly catalogued if at all, and gaining rights to the use of this footage can be prohibitively expensive. Lastly, some systems—Dustin (Ohmaye 1992) and Sickle Cell (Bell and Bareiss 1993)—have videotaped their own dramatizations in order to give students the feel of firsthand experiences. In all of these approaches, the cost of acquiring stories is significant, and the process can be time-consuming and difficult.

Adding to the cost of initial acquisition is the cost of transferring a story to a medium that affords a computer instant, random-access to individual stories. These media, in turn, place a limit on the amount of video that can be stored. The popular laserdisc format places 60 minutes of analog video footage on one side of a disk. This is the format used by Creanimate.

Digital video compression formats, in contrast, allow one to place video directly on a mass-storage device connected to the computer. Current compression algorithms, such as the MPEG1 standard, achieve compression rates of up to 50:1, a reduction from 27 megabytes of storage per second to 33 megabytes per minute (Cole 1993). Digital video compression promises to make significantly larger quantities of video available to computer-based learning environments at lower prices, but they will not entirely remove the cost or memory limitation considerations from the selection of stories for case-based teaching systems.

The costs and difficulties of acquiring stories and the limitations of storage media require that needs for stories be prioritized. The opportunities for learning that can arise in a task environment should be surveyed and ranked for educational importance. Once these priorities have been established, it is necessary to develop a strategy for collecting stories.

Story Selection Strategies

There are two approaches to the selection of stories for a storyteller, a top-down and a bottom-up approach. Generally, a mixture of the two is necessary for successful story collection. The top-down approach starts with an analysis of the learning opportunities in the task environment and a specification of the stories that suit those opportunities. The collection process then tries to meet those story specifications by drawing from a story source. The bottom-up approach starts with the story source and, after identifying good stories, finds or creates opportunities in the task environment to take advantage of those stories. In the extreme, the top-down approach consists of identifying every possible opportunity to tell a story in the task environment, creating a specification for a story to be told in that situation, and then going out and collecting all of the stories that meet one or more of those specifications. The bottom-up approach in the extreme consists of going to the story sources, collecting as many good stories as can be afforded and then either fitting them into the task environment or designing the task environment around them.

For a system of significant size, generating a complete top-down plan is too costly, but collecting stories bottom-up runs the risk of a story library that does not provide appropriate coverage. Therefore, a mixed approach is usually best. A mixed approach starts with a partial top-down plan that ensures adequate coverage of the task environment but allows for enough flexibility to take advantage of good stories that are not included in the top-down plan.

Depending on the source of stories, the amount of control the developers have over the availability of stories can vary greatly. A system in which stories are collected by interviewing experts lends itself better to the top-down approach then a situation in which the stories are collected from a library. In an interviewing approach, interviews can be structured to "go after" the desired stories, whereas in a system in which stories are drawn from a preexisting archive, stories specified by a top-down plan may not be available at any cost. However, as with other forms of knowledge acquisition, interviewing for story collection is an unpredictable process and does not necessarily produce the anticipated results. Thus, a top-down approach is more effective in situations in which the developers have control over the story creation process. On the other hand, in searching a large archive, as was done in Creanimate, a top-down plan can help to direct the search through the library and can provide a partial criterion for story selection. In both cases, the top-down plan should not unnecessarily exclude the selection of unanticipated but valuable stories.

Criteria for Effective Stories

When selecting stories for a storyteller it is important to insure adequate coverage, but it is also necessary to make sure the stories that are selected will be effective. Several stories may convey the same lesson but differ greatly in their effectiveness. Therefore, an important consideration for story selection is effectiveness. The three critical determinants of effectiveness are relevance, clarity, and memorability. The first is relevance:

Relevance. A story should relate closely to the opportunities for learning that arise in the task environment.

Stories can vary in their applicability to the situations that arise in the course of using a task environment. The more directly a story applies to an opportunity for learning that arises in the task environment the better. In addition, a story that contains lessons that apply across a range of opportunities for learning can be more valuable than one that has only a single application. It is important to recognize that relevance does not require that a story be about the particular situation in the task environment in which the story will be presented. The story need not be *about* that situation; it should simply be about a situation in which the same lesson applies. In fact, a system that tries to include stories that are *about* actual situations that arise in the task environment—as opposed to stories that are *relevant* to situations that arise—loses the efficiency of teaching with stories. Teaching with stories is designed to capitalize on the ability of people to reason from similar cases. This native ability enables us to build case-based teaching systems in which stories cover the domain by having relevance to a class of situations that arise in the system, not just one situation. If one were to try to get stories that are actually about all the situations that arise in the task environment, then he would end up with a one-to-one ratio between opportunities for learning and stories. This sort of design will not scale-up past a small library of stories, and will require either limiting the range of the task environment or only capitalizing on a small subset of the opportunities to present stories.

So, to be relevant a story does not need to be about particular situations that arise in the task environment. It need not even be about a situation within the domain being taught. The sole important criterion for relevance is: Does the lesson of the story apply to the context in which it will be used in the task environment? For example, one of the most common dialogues that students engage in with Creanimate is a discussion of how to make some non-flying animal (e.g., a fish) able to fly. The first step is almost invariably adding wings, but then students must consider how else to change their animal so it will actually be able to stay aloft. In this situation, a relevant story is one that shows the relationship between wing size and body weight in determining flight worthiness. To meet this need, a story should not be about this particular situation, i.e., someone trying to make a fish able to fly. It need not even be within the domain, i.e., about how birds in the wild fly. It simply needs to demonstrate the lesson that a certain ratio between wing-size and weight must be maintained in order to generate sufficient lift. Thus, a story about a B-52 bomber may be just as relevant to this situation as a story about an albatross.

The second criterion for effectiveness is clarity.

Clarity. The lesson conveyed by a story should be expressed clearly.

The importance of clarity cannot be overemphasized. There are three factors that determine the overall clarity of a story. First, the details of the story must be clear. The important characters, locations, and events must be clearly depicted. Second, the lesson of the story must be clear. Relevant outcomes should be clearly associated with the factors that induced them. One of the keys to clarity is conciseness. Extraneous or ambiguous details in a story can be confusing and even misleading. Finally, stories should be relatively self-contained. They should not require a significant amount of background knowledge to understand. A student should be able to comprehend the story with only the minimal knowledge that he brings to the interaction and his understanding of his current situation in the task environment. In short, stories should be sought that are unambiguous and not overly embroidered with details that will distract from the relevant lesson.

The third criterion for story effectiveness is by the student

Memorability. A story should contain features that lead to retention.

As I argued in the last chapter, the details in stories cause them to be better integrated and retained by learners than other forms of instruction. However, other attributes of stories can also contribute to better retention. Stories that appeal to emotions and aesthetics have more impact and are more likely to be retained. The reactions that stories can engender include surprise, humor, fear, revulsion, excitement, and sorrow. Any one of these elements can increase the impact of a story. When a story elicits one of these responses, that triggers several processes. First of all, the presence of an emotional impact leads to increased attention to the story. This can motivate students to pay attention to material that they would find dull or uninteresting otherwise. Second, an emotional impact can increase the extent to which a hearer identifies with the story. Emotional content induces a student to project himself into the story more. Third, it adds an additional set of memory structures, those dealing with emotions, to which the hearer can connect the story in his memory. All three of these processes may lead to stronger and more numerous connections between the information in the story and the other knowledge structures in the learner's memory. This, in turn, results in better retention of the story and a greater likelihood that the learner will be able to access it in situations when it will be useful in the future. Therefore, it is important to get stories that are vivid, dramatic or emotionally charged, because these elements increase students' interest in, attention to, and retention of the information in stories.

When selecting stories, each one should be evaluated according to its relevance, clarity, and memorability. These attributes can be weighed against each other so that, for example, a story that is clear but not terribly memorable may be selected over one that is unclear but extremely memorable. Since the unclear story may mislead the student, it is better to risk weaker retention of a clear story then to include a story that may not only mislead the student, but may do so in an extremely memorable fashion.

To summarize the issues of story selection—the primary objective is to establish sufficient coverage of the opportunities for storytelling that arise in the task environment. Because there is always an issue of cost and availability in collecting stories, it is necessary to develop an appropriate strategy for acquiring stories. Typically, a compromise must be reached between a top-down plan that specifies desirable stories in advance, and a bottom-up approach that enables you to identify valuable stories opportunistically. Finally, stories must be selected based on their effectiveness.

Story Selection in Creanimate

Stories selected for use in Creanimate must cover the range of issues that come up in the course of designing animals in the task environment. This range of stories is determined by the range of animals that students may choose to modify and the modifications that students may select for those animals. The limitation of 60 minutes for Creanimate's video library forced a compromise in the system's design. While the idea of Creanimate would allow students to enter any animal and any change they wanted, with only 60 minutes of video footage available, we could not have guaranteed relevant video for arbitrary combinations of animals and modifications. Therefore, we chose to present students with an initial list of animals and modifications. Selecting from these lists insures a student a dialogue illustrated with ample and interesting stories. Students were still permitted to type in an arbitrary animal and modification; however, the result of doing so could be "hit or miss."

Story collection in Creanimate combined top-down and bottom-up approaches. The lists of available animals and modifications provided the starting point for the top-down portion of the story selection process. These lists partially determine the range of situations a student might encounter. In generating the list of animals to modify, we were influenced by two factors: goals for breadth of coverage of the animal kingdom and input from surveys of children we conducted. For example, it was important to include an animal from each of the classes of birds, reptiles, mammals, fish, and insects, but within those classes the choices were influenced by the animals that students expressed interest in. The list of modifications for animals was developed in a similar fashion. It was important that the list of modifications include an appropriate variety of physical features, actions, and behaviors. However, the list was also influenced by the sorts of modifications that students discussed in surveys. These lists provided guidelines that allowed us to specify videos that would be useful in discussions of these animals and modifications. To see how these specifications were generated, consider one animal, a frog, and one modification, a beak, taken from the list of animals and modifications presented to students in the current version of Creanimate. The specification of videos that cover a frog with a beak are displayed in table 1.

Table 1.–A list of videos that could appear in a discussion of a frog with a beak. These specifications derive from the dialogue cycle of the dialogue manager and from the reminding strategies. The dialogue cycle and reminding strategeies are described in Chapters 6 and 4 respectively.

Stories that show...

- Uses for beaks:
    - pecking with a beak
    - grasping with a beak
    - probing with a beak
    - ...
- Other ways animals do the things that beaks are used for:
    - pecking with something other than a beak
    - grasping with something other than a beak
    - probing with something other than a beak
    - ...
- Different types of beaks
    - long, thin beaks
    - bills
    - beaks
    - ...
- What frogs do with the mouths they currently have.
    - grasping prey
    - carrying eggs
    - ...

Stories were collected from the film libraries of both the National Geographic Society and the Encyclopaedia Britannica Educational Corporation. Neither of these film libraries was designed to be an encyclopedic archive of animal adaptations. Instead, their contents have been determined by the productions that happened to have been made or purchased by these organizations. For that reason, during actual story collection, we were often unable to find videos that met the top-down specifications provided by the lists of animals and modifications. In cases where there were too many gaps in the available footage to cover a particular animal or a particular modification, items had to be dropped from the lists and replaced with new items for which there was adequate footage. On the other hand, we frequently came across videos that were relevant to an item on our lists that we had not anticipated in our story specifications. In addition, we encountered many videos that were so dramatic or startling that we felt they had to be included simply on the basis of their memorability. Because of the flexibility of the reminding strategies in Creanimate, these "must-see" stories were almost always worth including. One "must-see" story that was not accounted for by our initial plan showed a cheetah in slow-motion pursuing and catching a gazelle. While we had no idea how this video would ever come up in our existing plan, we could not resist the desire to include it. In fact, this cheetah story came up quite frequently in discussions of dancing, which is one of the most popular modifications for animals. This story is retrieved by the Similarity-Based reminding strategy which is described in Chapter 4:

```
That reminds me of another video.  Cheetahs also have long
legs, but instead of having long legs to dance, they have
long, muscular legs to run fast.
(I have an awesome video about that.)
Would you like to see that?
```

The story selection in Creanimate was conducted in parallel with the development of the task environment. The options available in the task environment were determined through a combination of top-down planning and the bottom-up influence of story availability. The range of both the task environment and the storyteller were determined by the instructional objectives of the system, by surveys of children's preferences, and by the availability of good stories. While insuring that the educational goals were properly covered, the selection of stories was flexible enough to accommodate the failure to locate certain desired stories and the serendipitous discovery of other unanticipated stories.

3.2.2 Communication with the Task Environment

While the selection of effective stories is critical for a storyteller, these stories are wasted if the storyteller is not able to locate stories when they are relevant to the students' situation in the task environment. To locate stories, a storyteller must be able to keep track of the students' interactions in the task environment, it must have a well-organized library of stories, and it must have strategies for locating stories when they are relevant. These goals are accomplished through a communication pathway between storyteller and task environment, an appropriate indexing vocabulary for the library of stories, and reminding strategies for retrieving stories. These three issues are the subject of this and the following two sections.

Even though the storyteller and task environment are independently functioning modules, it is important that they be designed to work together effectively. The storyteller must be able to monitor the task environment for opportunities to present stories, and it must be able to intervene at those moments when it has a story to present. This requires effective communication between the two components. At regular intervals during an interaction, the task environment must provide the storyteller with a description of the student's current situation. This description must be expressed in a format that the storyteller can use to search its library for a story that will assist that student. This communication requires a language in which the task environment can describe the current state of the interaction and which the storyteller can use to search for appropriate stories from its storybase. Therefore, the communication language between the task environment and the storyteller is constrained on the one hand by the nature of the task and the possible student states and on the other hand by the story library and the vocabulary in which it is indexed.

Communication with the Task Environment in Creanimate

Communication between the task environment and the storyteller is essential for the storyteller to identify and capitalize on opportunities for learning. The following describe circumstances in which the Creanimate task environment and storyteller communicate to allow the storyteller to present stories when they are relevant.

Responding to student answers with examples: When a student answers a question in the course of a dialogue, the dialogue manager communicates to the storyteller the question asked, the student's answer, and an evaluation of how appropriate the student's answer is to the particular question. Depending on whether the student's answer is correct, incorrect, or partially correct, and on whether the storyteller has an appropriate story for that condition, the storyteller intervenes with a relevant story.

Responding to student requests for suggestions: When a student asks the computer to suggest answers to a question, the storyteller responds with a story that suggests an appropriate answer. In order to do this, the dialogue manager must communicate to the storyteller what the current question is and the fact that the student has requested a suggestion.

Incidental remindings. At certain points in a dialogue, the storyteller may intervene with an "incidental" reminding. Incidental remindings do not directly relate to the current dialogue, but they provide students with opportunities to observe valuable, related phenomena. Incidental remindings rely on the storyteller's ability to monitor the interaction for factors such as a student's recent choices, the dialogue's current topic, and recently told stories.

Dialogue Planning: When the dialogue manager is preparing to initiate a dialogue, it must verify with the storyteller that the storyteller has applicable stories. If the storyteller has no relevant stories for a dialogue, the dialogue manager will not initiate that dialogue. This communication takes the form of a request for remindings relevant to the animal and the topic of the candidate dialogue.

The key to this communication between storyteller and task environment in Creanimate is that both components share a single knowledge representation. The Creanimate system contains a single knowledge base that describes animals and their attributes, and both the task environment and the storyteller draw on that knowledge base. The dialogue manager uses the knowledge base to help determine which dialogues to conduct, to draw inferences about the characteristics of animals, to ask questions and to evaluate students' answers to those questions. The storyteller uses the knowledge base as its vocabulary for indexing stories, enabling the reminding strategies of the storyteller to take advantage of the same ability to make inferences about animals in searching for stories that the dialogue manager uses in the course of conducting dialogues. The communication between the task environment and the storyteller is accomplished using the concepts from the knowledge base. They communicate with each other by passing data structures representing these concepts back and forth. This shared representation enables the dialogue manager to describe the current situation in the same vocabulary that the storyteller uses to index its stories.

3.2.3 Indexing Vocabulary

Once the stories for a storyteller have been collected, they must be organized and labeled in a fashion that will allow the storyteller to find them when they are relevant. In accordance with the terminology of case-based reasoning, the labels that organize stories are called *indices*. An index must capture enough information about a story that the storyteller can identify situations to which the story applies. An index is not a complete representation of a story, nor is it necessarily the summary of a story. The role of an index is to describe when and how a story should be told. Therefore, an index may make no reference to the contents of the story but may instead describe the sorts of settings in which the story is appropriate. For example, a case-based teaching system that uses a simulation to teach firefighters might contain a story about a fire that climbed up through the walls of a building without any visible flames and burst out three floors above firefighters. The index for that story would not necessarily describe anything about the contents of the story. Instead the index might describe situations in which the story is relevant. In that case, the index would indicate that the story applies to situations in which a student acts as if a fire is under control without checking the walls. Through its communication with the task environment, the storyteller would be able to identify when the student firefighter is in the situation described by the index, and the reminding strategies would find and present this story.

As this example shows, the indices in a case-based teaching system should be sparse in their representation. This sparseness represents a significant advantage for case-based teaching systems over other intelligent tutoring strategies. It enables a computer to teach with stories without requiring that it be able to represent or reason about their contents completely. In the traditional Intelligent Tutoring Systems paradigm (Sleeman and Brown 1982; Wenger 1987), the teaching system must be able to reason within the domain of study at the level of a student who has successfully mastered the domain. Instead, it is sufficient for the storyteller in a case-based teaching system to be able to represent the opportunities for presenting stories that arise in the task environment. The stories can convey the complex knowledge to the student without the storyteller needing to represent or understand it. Since an index describes the situations for which a story is appropriate, the elements that make up an index should describe these situations. Some of the elements that can be included in indices are:

- Misconceptions the story corrects
- Advice the story conveys
- Principles illustrated by the story
- Situations the story contains warnings about

Each of these categories describes a set of situations in which a story would be relevant. The indexing vocabulary must provide a language that is capable of describing these situations. Developing an indexing vocabulary is the sort of knowledge representation effort that is at the heart of any artificial intelligence system. As with other knowledge representation tasks, the situations to be represented must be analyzed to identify important features and relationships. In the case of a storyteller for a case-based teaching system, the domain to be represented may include the subject area being taught, the structure that underlies the task environment, and the pedagogical goals of the teaching system. The indexing vocabulary must be expressive enough about each of these three areas to relate together the material being taught, the situations that arise in the task environment, and the instructional goals of the system in a way that tells the storyteller which situations are right to present a particular story. As a student interacts with the task environment, the information that is encoded in indices gets examined by the storyteller in the course of pursuing its reminding strategies.

The Indexing Vocabulary in Creanimate

Because the indexing vocabulary is a central focus of this research, it is the subject of Chapter 7. The indexing vocabulary was determined primarily by the instructional objectives of the system. The most important of these objectives was to help students learn the basic relationships that underlie animal adaptation, and to provide them with cases that they can use to reason about these relationships. These relationships are expressed in the questions that Creanimate dialogues focus on, such as:

Why have a physical feature?
What features are necessary to support an action?
What survival behaviors does an action support?

The indexing vocabulary describes stories with respect to the questions they illustrate. For example, a story that shows a woodpecker using its beak to peck would be indexed to indicate that it illustrates the questions in table 2.

Table 2.–Some of the questions and answers that a story about a woodpecker pecking could be used to illustrate.

Q: What can a beak be used for?	A: To peck wood.
Q: What features can be used to peck?	A: A beak.
Q: Why do animals peck?	A: To probe for insects.
Q: What is one way animals get their food?	A: Probing for insects.

To construct indices that describe these relationships, Creanimate draws from taxonomies of animals, physical features, actions, and behaviors. The elements of these taxonomies are associated with each other in indices according to the relationships in which they occur in nature. Thus, physical features are associated with the actions they support and actions are associated with the survival behaviors they support. For instance, the index for the woodpecker story described above would include the relationship *a beak in order to peck wood*, where *a beak* is drawn from a taxonomy of physical features, and *peck wood* is drawn from a taxonomy of actions, and the relationship that connects them is one between a physical feature and an action for which it is used.

3.2.4 Reminding Strategies

A reminding strategy is a procedure for identifying a story to tell in a particular context. While we usually think of being reminded as a passive event, i.e., something that happens to you, a reminding is actually the result of active processes monitoring the observations coming in from the outside world and matching those observations against stories in memory (Schank et al. 1990a; Schank et al. 1990b). Teaching requires a special set of procedures for retrieving relevant stories that serve specific pedagogical goals. The reminding strategies in a case-based teaching storyteller serve this function. They monitor the progress of a student in the task environment and use features that describe his or her situation to identify stories that can help the student to learn from that situation. Different case-based teaching systems will have different reminding strategies that suit the subject being taught, the nature of the task environment, and the educational goals of the system.

Just as with the indexing vocabulary, the reminding strategies for any particular task environment must correspond to the types of opportunities for learning that arise in the task environment. Reminding strategies should be able to retrieve stories that correct misconceptions, give advice or warnings, or provide examples at the moment that the student can profit from them. While there are many reminding strategies that are useful across a wide range of teaching settings, certain settings and pedagogical goals necessitate special-purpose reminding strategies. Therefore, specific teaching systems may require specialized reminding strategies. For example, Burke and Kass (1992) describe a system that uses a social simulation to teach the skills necessary to sell management consulting services. This system uses the following six reminding strategies:

- Demonstrate alternative plan.
- Explain other's plan.
- Reinforce plan.
- Warn about plan.
- Demonstrate alternative result.
- Warn about perception [expectation].

These strategies are clearly specialized for a system that teaches how to construct and execute plans. They focus on the process of developing a plan. These strategies differ significantly from those in the Creanimate system which center on the presentation of examples of animal design principles. Burke and Kass (1992) point out that their strategies differ from the reminding strategies in Creanimate because the reminding strategies in Creanimate focus on the result of constructing a design whereas those in GuSS focus on the process itself. From this example, we can see that reminding strategies differ with the subject matter and structure of the task environment. As case-based teaching systems are developed for new domains, new reminding strategies suited to those subjects and tasks will be developed in order to respond effectively to the opportunities for learning from stories that arise in those systems. Once a significant body of case-based teaching systems are available, it will become clearer which reminding strategies are general-purpose and which are restricted to particular settings.

Reminding Strategies in Creanimate

The reminding strategies in Creanimate are suited for a task environment that conducts a design discussion. They identify and present stories that illustrate design combinations that work. Since the domain of Creanimate is animal adaptation, Creanimate's remindings show combinations of physical features, actions, and behaviors that existing animals use to help them survive in their niches in the wild. The Creanimate storyteller relies on three primary reminding strategies:

Example reminding. The example reminding strategy retrieves stories that show example cases of the principle under discussion. In a discussion of how animals swim, an example reminding might show the use of webbed feet.

Similarity-based reminding. The similarity-based reminding strategy retrieves stories that help students to make generalizations. By presenting a story that shows something that is "similar but different" to a preceding story, the storyteller helps the student to generalize the information shown in the first story to an appropriate level of abstraction. Similarity-based remindings also generate curiosity by exposing students to new things. After showing webbed feet being used to help a duck swim, a similarity-based reminding might show a Malaysian flying frog using webbed feet to help it glide through the air.

Expectation-violation reminding. The expectation-violation reminding strategy capitalizes on stories that violate common expectations in order to widen students' exposure and and generate interest. In a discussion of how animals swim, an expectation-violation reminding might show a fish that can move around without swimming by flapping across mud flats.

These reminding strategies are discussed in detail in Chapter 4. They each serve the pedagogical goals of the system by retrieving stories that will help the student to learn from his situation in the Creanimate task environment. The reminding strategies take into account the design being worked on, the design question under discussion, the current step in the discussion, and the stories that the student has already seen. These characteristics of a student's current situation are used by the reminding strategies which examine the information recorded in indices in order to identify stories that are appropriate for that moment.

3.2.5 Presentation of Stories: Bridging

Once a storyteller has located a story that responds to a situation in the task environment, it must present the story in a way that will maximize the student's ability to learn from it. In particular, the storyteller should help the student to make the connection between the contents of the story and his current situation in the task environment. By connecting the contents of a story to the context in which the story is being presented, the storyteller can help the student to understand the point of the story. Drawing explicit connections between the story and the features of the context that make the story relevant can also help the student to index the story as a case in his own memory. Since stories in a case-based teaching system are currently stored in an unmodifiable state, the storyteller cannot change the way it tells a story for particular contexts. However, it can precede and follow a story with information that will help the student to understand the story's connections to the storytelling context.

The conversational device used by a storyteller to make these connections is called a *bridge* (Burke and Kass 1992; Schank et al. 1992). A bridge is an introduction to a story that highlights the features of the current situation that have led to the telling of the story. For example, a bridge to the story about the boy who cried wolf might say, "I know a story about a boy who called for help when he didn't really need it. Wait until you hear what happened to him." This bridge both draws the child's attention to the features of the surrounding context that led to the telling of the story, and it raises the child's curiosity about the outcome. Bridges can be used not only to introduce a story but also following a story.[4] Bridges help a student to understand the connection between the point of a story and the features that make the story relevant to the current circumstances. This information enables the hearer to index the story in his own memory so that he can use the lesson it contains to help him understand similar situations in the future.

Bridging in Creanimate

In Creanimate, bridges serve three goals: they relate stories to the contexts in which they are presented, they help students to use context to index stories, and they advertise stories to students giving them information to decide whether or not to see a story. As an example, consider the following bridge:

```
Let's look at some things that other animals use to dance.
Maybe you'll want one of them for your bear.
Gulls dance.  Gulls use their legs to help them dance.
This is a funny video.
Do you want to see that?
```
(Transcript no. m2-4-28-11.54)

[4]For an example of how bridges can be used following a story, see the discussion of *codas* in Burke and Kass (1992)

This bridge is composed of two different parts. The first part makes the connection to context. It tells the student what the story has to do with the current discussion. In this example, the bridge says that the storyteller has found a video that shows a physical feature that animals use to dance. The second part advertises the video. It tells the student what the story is about, i.e., a gull using its legs to dance. The bridge also gives the student a subjective evaluation of the story. In this case it is funny; other stories are scary, gruesome, surprising, etc. Finally, the bridge shows the student a preview picture of the animal in the story. This picture is displayed in the lower righthand corner of the screen. In the event that the student is unfamiliar with the animal's name, this helps the student to know what he or she is being offered. In this way, bridges in Creanimate relate the story to the ongoing discussion, preview the relevant action in the story, and give the student some idea of the story's nature.

An important device that Creanimate uses in bridges is a question. Instead of telling the student what the story shows, as in the bridge above, Creanimate introduces a video with a question. So, instead of saying, "Gulls dance. Gulls use their legs to dance," it might say, "Gulls dance. Do you know what gulls use to help them dance?" The student has the chance to try to answer the question or watch the video to find the answer. If the student decides to watch the video, Creanimate follows up by repeating the question. Using a question as a bridge helps to make students more active viewers of stories. Rather than knowing in advance what they are going to see, they watch the story looking for the answer to a question. This not only increases their interest in the story, it helps them to focus on the aspects of the story that are relevant for the current discussion. Questions as bridges help to make the student a more active viewer and to focus on the relevant issues for his own animal.

3.3 Combining a Task Environment with a Storyteller

While each of the two components that make up the case-based teaching architecture has value on its own, together they provide a learning environment that is greater than the sum of its parts. The task environment and the storyteller in combination provide mutual support for the knowledge structures the student constructs in his memory. The task environment offers the student the opportunity to learn through direct experience. The storyteller provides the student with secondhand experiences. Each module is able to provide a form of learning that the other cannot. Because the task environment is an active learning environment, a student is able to learn skills and processes by doing them. These forms of expertise can only be properly learned by executing them. They can not be learned simply by having someone tell you how to do them, no matter how good a storyteller or explainer the instructor is. On the other hand, while skills and processes can only be learned effectively from experience, it can be frustratingly difficult if not impossible to discover the explanations and principles that accompany skills and processes in true expertise. One role of a storyteller is to provide this nonprocedural knowledge. A few words of explanation at the right moment can be worth more than hours and hours of personal experience for developing an understanding of causes and relationships. In addition, unguided learning-by-doing can be aimless and inefficient, even completely unproductive. The guidance that comes from hearing a story in the right context can prevent a student from wasting unnecessary effort in order to learn from experience. Experiencing all the situations necessary to learn a subject effectively can be frustrating and terribly inefficient. Therefore, a storyteller can greatly increase the breadth of learning that a student can receive from a small number of experiences.

The requirement of authenticity means that the structure of the task environment will reflect the structure of the subject matter being taught. The result is that a student's interactions with the task environment will help him to discover the underlying structure of the domain under study. This structure or framework, in turn, provides an organization that the student can use to store his new knowledge in his memory. Thus, the task environment helps the student to develop a framework for his new knowledge. When the storyteller recognizes an opportunity to present the student with a story, it intervenes with a case that will help the student learn from his situation. In this way, the storyteller provides the student with additional cases that increase his understanding of the subject matter. The student is able to use the framework that he learns from experience with the task environment to organize his memory of the cases that he learns from the storyteller. The framework helps the student to organize his new knowledge, and the cases act to strengthen and refine that framework. The organizational structure from the task environment provides structure for the cases from the storyteller. Likewise, the cases elaborate and support the structure from the task environment. The two elements combine to give the student a fuller understanding than he could get from a storyteller and a task environment independently.

To summarize, the task environment and storyteller are able to teach in combination what neither can teach independently. The task environment teaches skills and processes through activity. The storyteller provides explanations and relationships to support those skills and processes. The storyteller also gives a breadth of experience, albeit secondhand, that would be beyond the capabilities of a reasonable task environment. Finally, the task environment and storyteller help the student develop mutually supporting knowledge structures in his memory. The task environment provides structure and the storyteller provides cases. The structure serves to organize the student's understanding of the cases while the cases support and elaborate the knowledge encoded in the structure.

3.4 Summary

In this Chapter I have presented the case-based teaching architecture. Based on the principles of *active learning* and *learning from stories*, the case-based teaching architecture consists of two interdependent components. The task environment provides the student with an engaging interaction rich with opportunities for learning. The storyteller responds to opportunities in the task environment with stories that help a student to learn from his situation. The storyteller presents stories to students at the best possible moment, when they can understand and use the information contained within them. This supports Papert's (1980) *power principle* which states that information should be provided to a student in order to empower him or her to achieve a personally meaningful goal.

The task environment must be designed to allow a student to be an active learner. The student must be able to form and explore hypotheses in the context of a task that is both meaningful and relevant to the student's interests. In the course of interacting with that environment, the student must be able to experience interesting expectation failures that allow him to learn from his experiences. An effective task environment will provide a motivating interaction that includes appropriate quantities of challenge, relevance, control, curiosity, fantasy, and satisfaction. Finally, the task environment must be authentic to both the interests and goals of the student and to the legitimate use of the lessons learned in the real world.

The storyteller must be able to respond to opportunities for learning that arise in the task environment. It should be able to find and present stories to the student when they are relevant to his situation in the task environment. This requires that the storyteller have a sufficiently broad story library, indexed in a way that enables it to locate stories when they are appropriate, and it must have strategies for finding and presenting these stories. These requirements present five issues for a storyteller: story selection, communication with the task environment, indexing vocabulary, reminding strategies, and bridging.

Case-based teaching draws an important part of its effectiveness from the mutually supporting combination of task environment and storyteller. The task environment helps a student to understand the structure of a domain while the storyteller provides him with cases that go in that structure. Thus, the structure from the task environment and the cases from the storyteller support each other in the student's understanding of the domain.

Chapter 4

Reminding Strategies

As we go about our daily routines, we are constantly reminded of things that help us to accomplish our goals. In the course of solving problems, we get reminded of solutions to similar problems we've encountered in the past. We use these previous cases to help us solve our current problems. Similarly, in conversation we get reminded of experiences that bear on the current topic. When this happens, we usually find ourselves telling stories about those experiences (Schank 1990). Teachers in the classroom are no different. They also get reminded in ways that help them to accomplish their goals. They are frequently reminded of stories and examples that help them to convey their lessons. While we usually think of being reminded as an unplanned, inadvertent act, it is actually the result of active processes continually scouring memory for prior experiences that will shed light on a current situation (Schank et al. 1990a; Schank et al. 1990b; Schank 1982). In other words, remindings are an important outcome of the case-based reasoning process. The goal of this research is to harness the reminding process in order to teach with stories.

Different sorts of remindings are appropriate for different contexts. For instance, confirmation remindings are common in conversation. In a confirmation reminding a person demonstrates that he has understood a point that has just been made by telling a story of his own with the same point. Another type of reminding, counter-example reminding, is useful in an argument. Similarly, remindings that illustrate opportunities or warnings are valuable when developing plans. Each type of reminding requires a different strategy for retrieving relevant structures from memory. The reminding strategies that are useful for teaching are the subject of this chapter.

While individual teachers undoubtedly have their own idiosyncratic reminding strategies, general-purpose strategies are likely to be shared widely by teachers. The situations in which teachers use remindings and the categories of remindings that they employ constitute an important area for study. In the approach advocated by Braitenburg's (1984) "law of uphill analysis and downhill development," systems such as Creanimate and GuSS (Burke and Kass 1992; Kass et al. 1992) are exploring reminding strategies through the construction of systems that use them to achieve their goals. Braitenburg argues that it is easier and more productive to investigate a mechanism by attempting to build it rather than by analyzing it. He argues,

> A psychological consequence of this [law] is the following: when we analyze a mechanism, we tend to overestimate its complexity. In the uphill process of analysis, a given degree of complexity offers more resistance to the workings of our mind than it would if we encounter it downhill, in the process of invention. (Braitenberg 1984, 20-21)

In accordance with this philosophy, we are investigating reminding strategies for educational purposes by starting with educational goals and developing strategies to retrieve stories that achieve those goals. Empirical research will show whether these remindings are consistent with the strategies employed by human teachers.

Creanimate's reminding strategies fall into two categories, *example remindings* and *incidental remindings*. (The variety of example and incidental remindings employed by the Creanimate storyteller are displayed in table 3.) Example remindings display examples of the issues that arise in the course of a discussion. Example stories provide students with cases for their personal case libraries of animal adaptations. The cases provided by example remindings reinforce the framework of the domain that students learn from the explanation questions asked by the dialogue manager. There are three types of example remindings, each of which corresponds to one type of response a student might give to an explanation question. Specifically, students can give correct answers or incorrect answers, or they can request that the computer suggest an answer.

Table 3.–The Reminding Strategies in Creanimate

Example Remindings	**Incidental Remindings**
Correct Answer Remindings	Similarity-based Remindings
Suggestion Remindings	Expectation-violation Remindings:
Identical Remindings	Only-rule Remindings
	No-rule Remindings
	All-rule Remindings

In addition to example remindings, there are two types of incidental remindings. They are called "incidental" because they digress briefly from the central point of the discussion in order to present interesting sidelights. These sorts of digressions are, in fact, characteristic of good teachers. Good teachers are able to broaden students' exposure and excite them without sacrificing the main point of a discussion. The two types of incidental remindings generated by the Creanimate storyteller are similarity-based remindings and expectation-violation remindings. The similarity-based reminding strategy is designed to retrieve a story that is similar to an immediately preceding story but is also different from it in an interesting way. Similarity-based remindings help students to draw appropriate generalizations, and they introduce them to things they might not see in other contexts. The second form of incidental reminding, the expectation-violation reminding, retrieves stories that violate students' likely expectations. This reminding strategy uses the element of surprise to introduce students to uncommon phenomena. While the example reminding strategies provide the foundation for the discussion of explanation questions, the incidental reminding strategies pepper the discussion with relevant, interesting digressions. Together they provide the student with a dialogue that resembles a natural human discussion.

4.1 Example Remindings

The bread and butter reminding for Creanimate, just as it is for any human teacher, is the example reminding. The idea behind example remindings is simple. Instead of discussing an issue in the abstract, it is always better to discuss it in the context of a concrete example. Concrete examples are not only clearer and less ambiguous than abstractions, they support the natural process of case-based reasoning by providing cases. Consider the following transcript that has had the example remindings removed:

```
Suppose you could create a new animal by taking an existing
animal and changing it some way.  What would you make?
```

Student: A frog that can fight.
That's a good idea, is there a reason why you want your frog to fight?
Student: So it can defend itself
That's a good idea. Many animals fight to defend themselves.
So your frog can fight to defend itself.
You can commit to that now, or we can look at some other reasons why animals fight and you can decide later.

Contrast that transcript with an actual student interaction with Creanimate:

Suppose you could create a new animal by taking an existing animal and changing it some way. What would you make?
Student: A frog that can fight.
That's a good idea, is there a reason why you want your frog to fight?
Student: So it can defend itself
That's a good idea. Bees fight enemies.
Would you like to see that?
Student: Yes
[VIDEO: "BEES EXPEL WASP"]
Bees are not the only animals that fight enemies. For example, jawfish fight enemies.
I have a good video about that. This has battles in it.
Would you like to see a video about that?
Student: Yes.
[VIDEO: "JAWFISH DEFENDS TURF"]
So your frog can fight to defend itself.
You can commit to that now, or we can look at some other reasons why animals fight and you can decide later.
(Transcript no. m4-5-11-11.01)

Even though a transcript cannot convey the true impact of the two dramatic videos that the student saw in this interaction, the difference between the two transcripts is striking. The first never departs from the abstract. It contains no concrete details which the student can connect to his personal or observed experience. The discussion is couched in terms of "fighting", "defending oneself", and "animals." It does not refer to any particular form of fighting, reason for defending oneself, or specific animals. In contrast, the transcript from the actual Creanimate system uses vivid examples to put concrete meaning behind the abstract relationship *fight to defend oneself*. The specific videos show the student two cases, each of which contains a very different technique for fighting. In the first video bees fight a wasp that had invaded their hive by swarming over it in large numbers and stinging it to death. In the second, a jawfish attacks another in an effort to steal its hole. (Jawfish conceal themselves in holes on the sea floor while they wait for prey to swim by.) In this particular video, the defending jawfish successfully fights off the invader through a series of aggressive, biting lunges. Both videos are extremely compelling.

These stories provide valuable cases for a student's personal case library because they show two very different fighting techniques. However, they are also valuable for the similarities that they convey. In both cases, animals defend themselves and their home territory against an incursion. This commonality is in itself an important lesson for students because it emphasizes the relationship between the action fighting and an important survival goal for many animals, maintaining a home and territory.

As this transcript illustrates, example remindings illustrate answers to the explanation questions that Creanimate poses. The type of example the storyteller presents depends on a student's response to the explanation question. If a student answers correctly, the storyteller will present examples that confirm the answer. If a student answers incorrectly, the storyteller will show an example in which the student's answer appears. Finally, if a student chooses not to try to answer the explanation question but asks the computer to suggest answers instead, the storyteller will present example stories that suggest correct answers. These three types of example remindings are called *correct answer remindings, incorrect answer remindings,* and *suggestion remindings* respectively. There is, in addition to these three primary forms of example reminding, a fourth form called *identical reminding* which presents follow-up examples to accompany those produced by the primary example reminding strategies.

4.1.1 Correct Answer Remindings

A correct answer reminding is a story that reinforces a student's correct answer[1] to an explanation question. It confirms the hypothesis presented by the student with a concrete example. In the following transcript, the dialogue manager asks the student an explanation question of the category *Why feature?* in response to his request for a bear with a big nose.

```
...What would you make?
Student:   A bear with a big nose.
If your bear is going to have a big nose, that should help it
to do something.  Why would you like your bear to have a big
nose?
Student:   So it can smell good
```

---

[1] The use of the terms *correct* and *incorrect* is not really appropriate here for two reasons, one theoretical and one pragmatic. First, because Creanimate deals with a hypothetical world, there is really no such thing as an incorrect answer. Second, Creanimate's knowledge base is not capable of representing negative information. Creanimate is not capable of determining that an assertion is untrue, simply that there is no evidence that it is true. Therefore, *verifiable* and *unverifiable* are more appropriate descriptions of Creanimate's ability to evaluate students' answers. In Creanimate a "correct answer" is one for which the system is able to find an example in its knowledge base. Any other answer is unverifiable. The program's response to unverifiable answers, "I don't know any animals that...", is intended to communicate this fact. The terms *correct* and *incorrect* are used rather than *verifiable* and *unverifiable* for simplicity.

```
That might work.  Tapirs have big noses to help them to
smell.
This next video is really hilarious.  I like this one because
it has strange animals in it.  Is that something you would
like to see?
```
Student: yes

[VIDEO: "TAPIR SMELLS"] (Transcript no. Pal-m1-6-22-10.32)

In this case, the student proposes that his bear have a big nose in order to "smell good." Since the knowledge base does not contain a concept for either "smell good" or "smell well" (which is what I assume the student meant), the parser interprets the student's answer as the action *smell*. After parsing the student's input, the next step is the evaluation of the student's answer by the dialogue manager. (This evaluation is described in Chapter 6, The Dialogue Manager). Once the dialogue manager ascertains that the student's answer is correct, that is, that there are animals that use big noses in order to smell, the storyteller initiates the correct answer reminding algorithm. This algorithm, described in the next chapter, identifies stories that contain examples of animals using big noses to smell, i.e., the "Tapir Smells" story. Correct answer remindings like the one above both reward students for correct answers and provide them with cases that support their hypotheses.

4.1.2 Incorrect Answer Remindings

The second example reminding strategy is the incorrect answer reminding strategy. Incorrect answer remindings are generated when students propose answers to questions that the dialogue manager is unable to verify. Incorrect answer remindings follow both explanation questions and bridging questions[2]. An incorrect answer reminding shows the student a story in which his answer does appear. In fact, an incorrect answer reminding always shows a question for which the student's answer would be correct. Consider the following transcript:

```
OK, Let's find a reason for your bear to have a beak.  Is
there a reason you want your bear to have a beak?
```
Student: So it can smell better

```
I've never heard of any animals that have beaks to smell, but
there are some other ways that animals smell.
For example, pikes smell.  Pikes use their nostrils to help
them to smell.
I have an interesting video about that.  I like this one
because it has mean fish in it.
Would you like to see a video about that?
```
Student: yes

[VIDEO: "Pikes smell"] (Transcript no. m4-5-8-11.37)

[2]Bridging questions are described in Section 7.3.4.

In this transcript, the dialogue manager asks the student for a way to use a beak. The student proposes smelling. Since the dialogue manager is unable to find evidence that animals use beaks to smell, it considers the student's answer unverifiable. In response, the incorrect answer reminding strategy retrieves a story that shows one way that animals *do* smell. It retrieves a story that shows pikes using their nostrils in order to smell. In other words, it finds an example of a question for which the student's answer would be correct response, i.e., "Why have nostrils?"

The goal of the incorrect answer reminding strategy is to provide students with stories that enable them to learn from expectation failures. In answering questions, students propose hypotheses. When a hypothesis proves incorrect, the student has an opportunity for learning. In fact, the student is in a position to learn two things. In the transcript above, the student can learn 1) correct reasons that animals have beaks and 2) actual ways that animals smell. The student will have the opportunity to learn the former, the correct reasons, later in his dialogue as he continues with the discussion of why his bear should have a beak. The second opportunity for learning, actual ways that animals do smell, is exploited at the moment the student gives the unverifiable answer, by the incorrect answer reminding strategy. In this way, the incorrect answer reminding strategy capitalizes on incorrect answers as an opportunity for students to learn from the failure of their expectations.

The following transcript shows another example of an incorrect answer reminding. In this transcript, the girl has created a mouse with wings so that it can fly. The next question is how it will keep itself up in the air, and she suggests that the mouse be larger.

Ok, your mouse will have wings so that it can fly.
In order to fly, animals must be able to keep themselves in the air. Let's talk about how your mouse will keep itself up in the air.
Having wings is not enough for your mouse to keep itself up in the air. It needs more.
How else do you want to change your mouse so it can keep itself up in the air?
Student: Give it a larger body or large wings or something that will keep up in the air
I did not understand that. How do you want to change your animal?
Student: Give it a larger body than it already has
I don't know of any animals that have large so they can keep themselves in the air. But, I do know some other things that large are used for.[3]
For example, lions use their large size to help them threaten other animals.
Would you like to see that?
Student: Yes

---

[3] This output from an early version of Creanimate contains grammar mistakes that were subsequently corrected.

[VIDEO: "HYENAS AND LION"]
That's the only thing that I know of that large are used for. Let's return to the task of getting your animal to keep itself up in the air. (Transcript no. m3-5-11-15.~34)

The student's request for a larger body will not help her mouse to fly. In fact, it will make it harder. However, since the student has raised the topic of large body size, the storyteller capitalizes on this as an opportunity to show an interesting story that shows something that large size is good for. The assumption behind incorrect answer remindings is that at the moment a student suggests an answer to a question, the student will be interested in learning something about the answer regardless of its correctness. The incorrect answer reminding strategy gives a student the opportunity to observe and learn something about the answer that he provided even when it is incorrect for the question at hand. The incorrect answer reminding strategy promotes the idea that incorrect answers should not be treated as embarrassments to be punished, but as what they are, opportunities for learning. Correct or incorrect, a student has an investment in his or her answer and will be motivated to see it in a story and learn more about it.

4.1.3 Suggestion Remindings

The third form of example reminding is the *suggestion reminding.* These remindings use stories to suggest answers to explanation questions. In Creanimate, students always have at least two options for responding to an explanation question. A student may offer an answer or he may ask the program to suggest answers. If a student asks for suggestions, the computer presents them in the form of example remindings. In the following transcript, the student asked for a dog with wings in order to fly. After making his dog smaller so it can keep itself up in the air, the dialogue manager raises the question, "Why fly?"

We need a reason for your dog to fly. Is there a reason you want your dog to fly?
Student: Give me some suggestions
Canada geese fly. Do you know why canada geese fly? (This next video is really good.)
Student: Show me a video
[VIDEO: "GEESE MIGRATE"]
Now, do you know why canada geese fly?
Student: So they can migrate
Yes, that is right. Canada geese fly to migrate from cold. So, a dog might fly to migrate from cold. Why would you like your dog to fly?
Student: Migrate from cold
OK, your dog will fly in order to migrate from cold.
(Transcript no. Pal-m1-6-5-12.57)

The Canada geese story is used to suggest an answer to the student. Like the other example reminding strategies, the suggestion reminding strategy helps the student to learn from concrete cases. In this transcript, the story suggests migration as a reason for flying. The program could have simply said, "One reason to fly is to migrate from cold," but suggesting answers through examples allows students to learn through the more natural process of learning from cases. Like incorrect answer remindings, suggestion remindings capitalize on the opportunities for learning that arise when a student becomes aware of a gap in his knowledge. In this case, the student may have asked for suggestions because he did not know any reasons for flying. Suggestion remindings respond to students' requests for information with concrete cases that fill gaps in their understanding.

4.1.4 Identical Example Remindings

In most situations, Creanimate will present more than one example reminding if it can find them. When the storyteller identifies more than one example reminding, it ranks them according to a rating of intrinsic appeal provided by the indexer, and it presents the highest rated reminding as the example reminding. If the student elects to watch the proposed example story, the storyteller will follow it with the other examples that it had identified for the current context. All the stories that are appropriate examples for a given context are called *identical remindings.* In the following transcript, a story about a katydid is followed by an identical reminding about a caterpillar:

> We need a reason for your frog to hide. Is there a reason you want your frog to hide?
> **Student: So it can not get eaten**
> That might work. Katydids hide to avoid detection by predators.
> I have a cool video about that. This video is for people who like surprises.
> Would you like to see that?
> **Student: Yes.**
> *[VIDEO: "KATYDID MIMICS"]*
> Katydids are not the only animals that look like parts of plants in order to avoid detection by predators. For example, caterpillars hide to avoid detection by predators.
> This is a surprising video.
> Would you like to see a video about that?
> (transcript no. Pal-m2-6-13-10.15)

The katydid story is presented as a correct answer reminding to confirm that the student's reason for hiding, to avoid being eaten, is a good reason for hiding. The storyteller follows this with a second reminding about an animal that hides in order to avoid getting eaten. This caterpillar story is called an identical reminding. In any particular dialogue, the student can see as many identical remindings as the storyteller can find. The name *identical reminding* can be misleading; two identical remindings do not necessarily depict identical events. For example, if a student answers the question "Why do you want your animal to fly?" with "so it can hunt," two identical remindings might be

1) an eagle flying to search for its prey and
2) a hawk flying in pursuit of its prey.

Even though searching for prey and pursuing prey are not identical behaviors, these are considered identical remindings because both searching for prey and pursuing prey are part of hunting. Both are examples of *flying in order to hunt*. Since an example may be more specific than the thing it exemplifies, two identical remindings may be examples of the same concept without being identical to each other.

Identical remindings are also offered when a student chooses not to see a story proposed by the storyteller. When Creanimate retrieves an example reminding, it introduces the story with a bridge and asks the student if he wants to see it. If the storyteller retrieved other, identical remindings, then the student is also given the option to see one of them. This option appears to the student as the choice, "What other videos can I see?" when he is deciding whether to see the suggested video or not. For example, in the transcript above, when the storyteller proposed the story about the katydids, the student saw the following:

```
That might work. Katydids hide to avoid detection by
predators.
I have a cool video about that. This video is for people who
like surprises.
Would you like to see that?
  Yes.
  What other videos can I see?
  No.
```

Following the principles established in Chapter 3, the student is given as much control over his learning interaction as possible. Whenever possible, Creanimate allows the student to select an alternative over the story proposed by the storyteller. In addition, through bridges, it gives the student as much information as possible to make the decision whether or not to see a particular story.

4.1.5 Bridges for Example Remindings

Once the example reminding algorithm has identified appropriate stories, it selects the story with the highest rating to present as the example reminding. To introduce the story, the storyteller constructs a bridge that tells the student how the story relates to the current discussion. In the transcript below, the storyteller introduces a correct answer reminding by telling the student what the story shows.

OK, let's find a reason for your frog to have wings. Is there a reason you want your frog to have wings?
Student: So it can fly
<u>That's a good idea. White breasted sea eagles use their feathered wings to help them to fly.</u>
<u>Would you like to see that?</u>
Student: Yes
[VIDEO: "Sea Eagle Catches Fish"] (Transcript no. m2-4-27-14.35).

This bridge describes the specific type of wings that are used to fly in the story. The bridge is intended to give the student enough information to decide whether or not he wants to see this story, and to relate the story to the ongoing dialogue. In addition, the student sees a still picture taken from the story that shows the animal in the story. Figure 25 shows this.

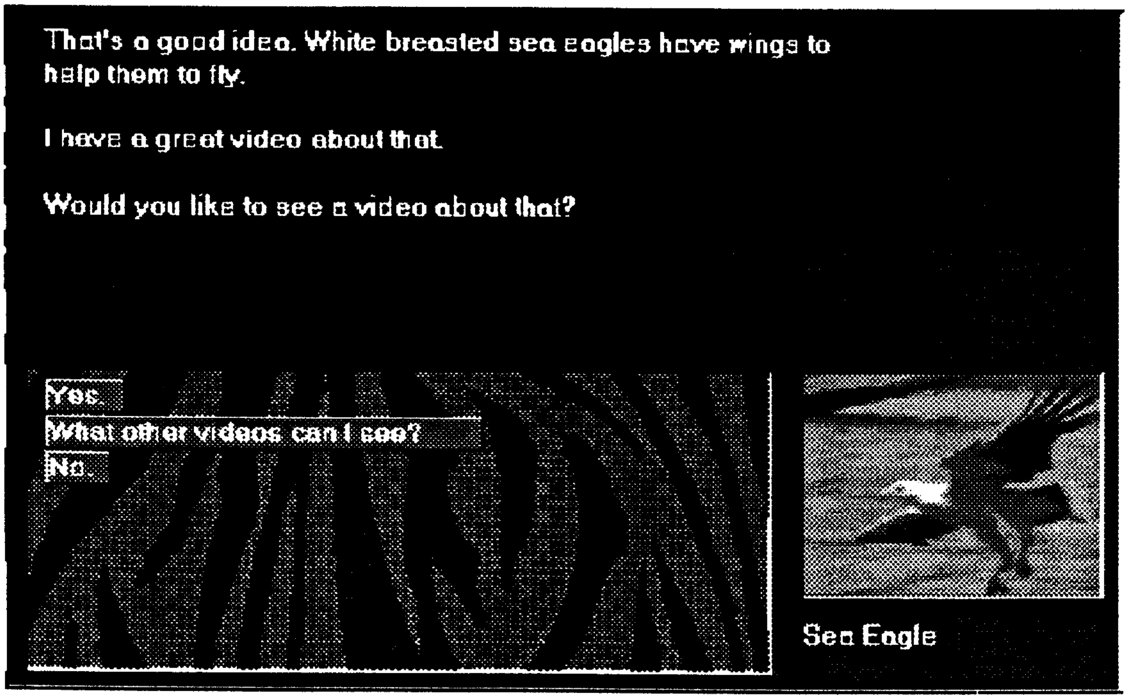

Figure 25. A portion of the Creanimate screen showing a bridge to a correct answer and the accompanying picture of the animal in the story.

Correct answer remindings are always introduced with statements that explain how the story is an example of the concept under discussion. However, the bridges for incorrect answer remindings and suggestion remindings often take the form of questions. The following transcript which also deals with an animal with wings shows a question used as a bridge:

> If your bear is going to have wings, that should help it to do something. Why would you like your bear to have wings?
> **STUDENT: Give me some suggestions**
> Sage grouses have wings. They have feathered wings. <u>Do you know what sage grouses use their wings for?</u>
> (I think this video is hilarious.)
> **STUDENT: Show me a video**
> *[VIDEO: "GROUSE DANCES"]*
> Now, do you know what sage grouses use their wings for?
> **STUDENT: So they can dance around**
> Yes, that is right. Sage grouses have wings to help them to dance. (Transcript no. Pal-m1-6-2-16.48)

The sage grouse story is introduced with the question, "Do you know what sage grouses use their wings for?" By introducing stories with questions, Creanimate helps make students active viewers. When a student is looking at a video with a question in mind, he will focus on the video more closely because he is looking for an answer to the question. This helps activate the student, and encourages him to attend to the aspects of the video that are relevant to the current discussion. Videos such as those of animals in the wild contain a great deal of information and can convey many lessons. By asking a question as an introduction, Creanimate provides a context that assists the student to focus on the particular elements of a story that apply to his animal. If the student attempts to answer the question before seeing the story, the dialogue manager will evaluate his response. If he is correct, he will get confirmation and then be offered the opportunity to see the story. If his answer is unverifiable, he will be offered an incorrect answer reminding before having the opportunity to return and view the current story. However, if the student chooses to see the story without trying to answer the question, as the student in the transcript above did, then the question will be repeated following the story. After the story, the student has the opportunity to try an answer, review the story, or ask the system to tell him the answer. In the example above, the student asked to be told the answer. Which stories should be introduced with questions is a decision made by an indexer as he adds information to indices.

Questions serving as bridges, like the one above, are an important element of Creanimate's strategy for providing stories in meaningful contexts. These contexts assist students to understand stories, retain them as cases, and index them appropriately in their own memories.

4.1.6 The Role of Example Remindings

Each type of example reminding is designed to help a student learn from the context provided by the discussion of his animal. As I stated in Chapter 3, the task environment in a case-based teaching system has two important goals: 1) help the student develop an understanding of the structure of the domain under study and 2) provide him with valuable opportunities to learn from stories . In the Creanimate system, explanation questions provide the structure of the domain for the student. Questions such as, "What physical features are necessary to fly?" or "What survival behavior can flying enable an animal to achieve?" communicate the basic underlying relationships of animal morphology to a student. These questions help the student to understand the relationships that connect an animal's physical features, actions and behaviors. These relationships provide the student with a structure to organize his knowledge about animal adaptation.

However, people rely on cases for reasoning about complex domains. Thus, structure is not enough; students also need cases. The purpose of example remindings is to provide students with concrete cases that reinforce the structure provided by explanation questions. These cases supply material for the natural process of case-based reasoning. The structure provided by the questions can be used by the student to index the cases that come from example remindings. The structure and cases therefore work together as the student constructs an understanding of the domain.

4.2 Similarity-Based Remindings

The following transcript shows a similarity-based reminding. In this transcript, the student has just decided to give a bear wings so it can fan itself with them.

Let's see if we can find a reason for your bear to fan itself. Can you think of a reason for your bear to fan itself?

Student: So it can stay cool in summer

That might work elephants fan themselves to keep cool.

I have an interesting video about that. This has baby elephants in it.

Is that something you would like to see?

Student: Yes.

[VIDEO: "ELEPHANTS AT WATERHOLE"]

That last video made me think of another one. This one is funny. This has new ways to keep cool in it.

Meerkats also keep cool. Only, instead of fanning themselves, they do something different. They turn over sand.

Do you want to see that? (Transcript no. Pal-m1-6-3-22.42)

In the first part of this transcript, the storyteller shows a correct answer reminding in response to the student's suggestion that his bear fan itself in order to keep cool. This particular example story shows elephants flapping their ears as a way of cooling off. The telling of this elephant story triggers a similarity-based reminding of a similar, but different story about keeping cool. Rather than showing an animal fanning itself to keep cool, the similar reminding shows meerkats digging down to a cooler layer of sand to keep cool in the hot desert. (Meerkats, relatives of the mongoose, are small desert animals that resemble prairie dogs in their social behavior.) While this meerkat story is not strictly on point for the student's discussion of why his bear will fan itself, it reflects a common strategy employed by human teachers. The reminding constitutes a digression brought about by the new topic of keeping cool. The storyteller, like a good teacher, recognizes the introduction of a new topic as an opportunity to introduce the student to an interesting variation on the current theme that will broaden his exposure and help him to generalize his understanding beyond the specific examples he may have been exposed to before.

Similarity-based remindings are incidental remindings because they digress slightly from direct consideration of the student's animal. Digressions like these are inspired by the way human teachers take advantage of situations that arise in the course of teaching in order to bring in new ideas. In contrast to example remindings that capitalize on the opportunities for learning that arise from explanation questions, similarity-based remindings[4] exploit the opportunities that arise from telling other stories. After seeing an example reminding, a student can be expected to start making generalizations that will help him relate the new information in the story to his existing knowledge. By intervening at that moment with stories that are similar but not identical, the similarity-based reminding strategy is intended to assist the student in making those generalizations. Thus, the timing of similarity-based remindings is critical. A similar reminding follows directly on the heels of an example

[4] For brevity, similarity-based remindings are also called *similar remindings.* The terms are interchangeable.

reminding but before any identical remindings. Below is another example of a similarity-based reminding. This one appears in a discussion of why a student's butterfly might fight:

> `Thomsons gazelles fight for their mates.`
> `I have pretty good video about that. This has fast action in it.`
> `Would you like to see a video about that?`
> **`Student: Yes.`**
> *`[Video: Gazelles Spar]`*
> `That thomsons gazelle video reminded me of a hilarious video. Sage grouses compete for mates. Only, instead of fighting for their mates, they compete for the attention of a female.`
> `Is that something you would like to see?`
> **`Student: Yes`**
> *`[Video: Grouse Dances]`* (Transcript no. m4-5-4-14.41)

This transcript shows the use of similarity-based remindings both to help the student make a generalization and to broaden the student's exposure. The example story shows male gazelles fighting to compete for the right to mate. After seeing this story, the student will begin to integrate the new information it contains into his understanding of animal behavior. This presents an opportunity to show an additional story that will help the student to make new connections and form valuable generalizations. In the transcript above, the similarity reminding the storyteller presents will help the student to recognize that competing for mates is a general behavior pursued by many animals in many different ways and that fighting for mates is only one of these ways. The similar story presented shows a very different way of competing for a mate: male sage grouses do an elaborate mating dance as part of their efforts to attract a mate. Seeing these two stories in juxtaposition will help the student to recognize the generality of competing for mates and it provides him with two different cases that both support this generalization. In addition, this particular similar reminding is likely to broaden the student's range of experience. It is a good bet that the ritualized mating dances in this story show the student something that he has never seen before. In this way, similarity remindings play the important role of drawing students into new areas and awakening new interests. While they are interjections in conversations, they are natural ones because they always relate directly to a story the student has just seen.

4.3 Expectation-Violation Remindings

As I argued in Chapter 3, the greatest opportunity for learning takes place when an expectation is violated by observation or experience. This is what Schank (1982) calls failure-driven learning. The failure of an expectation prompts an individual to look for the cause of the expectation failure, initiating the learning process. However, expectation failures do not just trigger learning, they also provoke interest. In general, stories are interesting to the extent that they challenge our expectations rather than confirm them. After all, an experience that conforms exactly to expectation is boring. The expectation-violation reminding strategy is an incidental reminding strategy that uses expectation failures as a way of provoking interest and triggering learning. In order to generate remindings based on expectation-violations, the system must have knowledge about the expectations that students are likely to possess and about which stories violate those expectations. The following transcript shows an example of an expectation-violation reminding:

```
How would you like to change your pike?
Student:    A pike with wings.
Actually there are fish that have wings.  For example, manta
rays have fins that are like wings.
Would you like to see a video about that?
Student:    Yes.
[VIDEO:  MANTA RAYS SWIMMING]
```
(Transcript no. m1-4-24-11.32)

The manta ray story was retrieved by the expectation-violation reminding strategy. It violates the expectation that fish don't have wings. In order to present this story, the storyteller used the knowledge that a student would not expect a fish to have wings and that this story violates that expectation. This information is represented in indices using rules to encode *standard expectations*. A standard expectation is an expectation that an indexer believes students in the target population are likely to possess. These expectations are expressed as one of three types of rules:

Only-rules, e.g., "Only mammals have hair."
No-rules, e.g., "No mammals lay eggs."
All-rules, e.g., "All birds fly."

While the system treats the rest of the information in its knowledge base as true under all circumstances, the information expressed as standard expectations is used only for the purposes of representing likely student beliefs. The other information in the knowledge base is used to draw inferences about animals in response to students' actions. In contrast, standard expectations are only used for expectation-violation reminding.

The expectation-violation reminding strategy works in the following way. When the storyteller detects that a standard expectation applies to a student's current situation, it presents a story that violates that expectation. The intended result is that a student will develop more sophisticated expectations. In addition, expectation-violation remindings share with similarity remindings the goal of broadening a student's range of experience. Finally, the element of surprise that accompanies an expectation-violation serves a valuable motivational purpose. In expanding a student's horizons, the reminding strategy also increases their interest and curiosity.

The expectation-violation reminding strategies are divided into two categories. The remindings that result from the violation of only-rules and from no-rules form one category, and the remindings that result from the violation of all-rules form the second category. The two strategies are triggered by different situations in a student's discussion.

4.3.1 Only-Rule and No-Rule Remindings

The following two transcripts show remindings prompted by an only-rule and a no-rule respectively.

Only-rule: Only birds have beaks.

```
How would you like to change your butterfly?
Student:    A butterfly with a beak.
Actually there are animals other than birds that have beaks.
For example, octopi have curved beaks.
Is that something you would like to see?
Student:    Yes.
[VIDEO: "OCTOPUS SPAR"] (Transcript no. m3-5-1-11.33)
```

No-rule: No fish move through the air.

```
So, your pike could have wings to fly.
You can commit to that now, or we can look at some other
reasons animals have wings and you can decide later.
Student:    To fly.
Ok, your pike will have wings so that it can fly.
[Picture of a pike with wings]
Actually there are fish that move through the air.  For
example, salmon jump.
Do you want to see that?
Student:    Yes.
[VIDEO:  "Salmon spawn as bear watches"]
(Transcript no. m3-5-7-15.06)
```

Only-rule and no-rule remindings are incidental remindings that get triggered when a student takes an action that changes his animal. These remindings can occur when the student selects an initial modification for his animal or when he resolves an explanation question by committing to a new modification for his animal. In the first example above, the reminding is triggered by the student's initial request for a butterfly with a beak. In the second, the reminding is triggered by the student's decision to commit to flying as a reason for his fish.

The first reminding is triggered by an only-rule that says, "Only birds have beaks." This is a standard expectation that is violated by the octopus, which has a large, curved beak concealed beneath its body. Since the student's animal, a butterfly with a beak, also violates this expectation, it triggers the only-rule reminding. The second reminding is triggered by a no-rule. The student's request for a pike that has wings in order to fly violates the standard expectation, "No fish move through the air." In fact, salmon do move through the air when they jump over rapids and waterfalls during their spawning runs up streams and rivers. This expectation-violation triggers the reminding. Prompted by a pike that flies the storyteller gets reminded of salmon leaping through the air.

To summarize, both no-rule and only-rule remindings are triggered by students' changes to their animal. When a student's animal and an animal in a story both violate the same standard expectation, the expectation-violation reminding strategy causes the storyteller to be reminded of the story.

4.3.2 All-Rule Remindings

All-rule reminding, like similarity-based remindings, follow directly on the heels of example stories. They are triggered by the contents of the example stories they follow.

Unlike only-rule and no-rule remindings which respond to an animal that violates standard expectations, all-rule remindings respond to example stories about animals that *conform* to standard expectations. In response to a story that shows something that agrees with standard expectations, an all-rule reminding will show something that violates those same expectations. In the following example, a student is working on a frog that can fly. Predictably, the student asks for wings. Since wings used to fly conforms to a standard expectation, an all-rule reminding produces an example of wings being used for something truly surprising:

```
Ok, what should we give your frog to help it fly?
STUDENT:  give  it  wings
I like that idea. White breasted sea eagles use their large
wings to help them fly.
I think this video is great.
Do you want to see that ?
STUDENT:  yes
[VIDEO: Sea Eagle Catches Fish]
Birds don't just use wings to fly. For example, black herons
use their wings to make shade.
Would you like to see that?
STUDENT:   yes
[VIDEO: "heron shades water"]
```

In the *heron shades water* story, the student sees a heron standing in a pool of water with its wings spread to create a shady area. Fish looking for relief from the heat of the sun are attracted to the shade where they become easy prey for the heron. This unusual use for wings typically comes as a surprise.

All-rule remindings of this sort serve two primary purposes. First, the surprise that ensues from them operates as a motivator. Students enjoy seeing the unexpected, so these surprising remindings enhance their pleasure in using the system. Second, these remindings widen students' exposure. Through these all-rule remindings students have an opportunity to see some lesser known phenomena from the animal kingdom that they might not encounter any other way. Because of the context provided by the ongoing discussions of their animals students take interest in these stories in a way they might not otherwise. Ideally, these remindings lead students to become curious about the phenomena they see and encourage them to use Creanimate as well as other resources to satisfy this curiosity.

4.3.3 Standard Expectations

In order to produce expectation-violation remindings it is necessary to represent standard expectations. Standard expectations are those that we can reasonably expect students to possess. The criterion we use for identifying when a standard expectation is relevant is:

> If the contents of a story are judged to be surprising to the average adult, then the story contains a violation of a standard expectation.

The risk in this approach is that a story will be presented as an expectation-violation even though the student already is familiar with the exception in the story. By setting our threshold for expectation-violations reasonably high, i.e., surprising to the average adult, we minimize the frequency of an expectation-violation reminding not actually surprising a particular student. When this does happen, then, it means that the student is unusually sophisticated about the subject matter. Instead of this presenting a problem, we have observed that these remindings are perceived as a reward that reinforces the student's pride in his exceptional knowledge. Thus, while Creanimate is unable to judge what the student's actual expectations are, it nevertheless is able to present expectation-violations that consistently engage a wide range of students.

4.4 Summary

The Creanimate storyteller employs several reminding strategies, each designed to present stories that help students to learn from the animals they create and the resulting discussions. The primary reminding strategy in Creanimate is the example reminding strategy. The three primary example reminding strategies correspond to the three ways that students may respond to explanation questions. The correct answer reminding strategy provides examples to support a student's correct answer to an explanation question. The incorrect answer reminding strategy helps a student to learn from incorrect answers. It presents examples of the answer in a way that shows the student situations in which his answer would be correct. The third example reminding strategy, the suggestion reminding strategy, suggests answers to students when they don't know any or are too timid to try one. This strategy suggests answers through stories that show examples of them. Finally, the identical reminding strategy provides follow-up examples to each of the three primary example strategies. All of the example reminding strategies are designed to give students cases that support the abstract framework provided by explanation questions.

In addition to example remindings, the Creanimate storyteller employs two types of incidental reminding strategies. The first is the similarity-based reminding strategy which follows up example remindings with stories that show something "similar but different." Similarity-based remindings help students to form appropriate generalizations, and they expose students to new phenomena. Expectation-violation remindings, the second type of incidental remindings, also broaden students' exposure. They also take advantage of the natural motivation to learn that results from surprise. The three types of expectation-violation remindings are based on standard expectations. Standard expectations are expressed in the form of rules that represent the beliefs that students can reasonably be expected to possess. Only-rule and no-rule remindings respond to a student's change to his animal with stories about surprising animals that violate the same expectations that the student's animal does. All-rule remindings follow example stories that conform to standard expectations with surprising stories that contain exceptions to those expectations. All three forms of expectation-violation reminding are designed to increase student curiosity and interest through expectation failures.

While the example reminding strategies provide the foundation for a discussion of explanation questions, the incidental reminding strategies provide relevant, intriguing interjections. These incidental remindings use the current context to give students a wider range of experience and, hopefully, to pique their curiosity and interest. The combination of these strategies provides the feeling of a natural dialogue, complete with digressions, conducted by a human teacher with a wide range of knowledge about the subject matter.

All of the reminding strategies are designed to fulfill the educational objectives of the Creanimate system. They present stories in a way that is calculated to take advantage of a student's investment in creating his own animal and the context provided by explanation questions. Because the stories respond directly to the student's discussion of his animal and they help him to resolve pending issues for that animal, the student is motivated to learn from them. The context helps the student to index the contents of a story with respect to the issues in the current discussion. This indexing will help the student to retrieve the cases gleaned from stories when they become useful in the future. In addition, those cases help reinforce the structure of the domain that is provided by the explanation questions. Combined in this way, the structure provided by explanation questions and the cases from stories mutually reinforce each other in a student's memory.

Chapter 5
Reminding Algorithms

In the previous chapter I presented the reminding strategies in Creanimate from the perspective of their pedagogical roles. Pedagogy aside, the implementation of these reminding strategies is a research issue that falls squarely within the realm of artificial intelligence. Implementing reminding strategies on a computer poses a difficult challenge. It requires both an expressive indexing scheme to organize stories in the computer's memory and sophisticated algorithms for retrieving stories when they are relevant. In this chapter, I describe the algorithms used to implement each of the reminding strategies described in the previous chapter. For each strategy, I describe both the indexing information required to generate remindings of that type and the algorithm used by the storyteller to produce the remindings.

5.1 The Indexing Vocabulary: An Overview

Before we can look at the reminding algorithms themselves, it is important to understand the memory structures on which they operate. The reminding algorithms in a case-based teaching system translate a characterization of the current state of the task environment into a description of stories that are appropriate for that situation. In Creanimate, this description is constructed using elements drawn from the indexing vocabulary. This is the same vocabulary that an indexer uses to index the stories in Creanimate's story library. To understand the reminding algorithms, some familiarity with the indexing vocabulary is essential. (This vocabulary is discussed in detail in Chapter 7.)

The Creanimate indexing vocabulary is composed of two classes of representational structures, objects and relationships. Objects are used to represent individual concepts such as animals, features, and actions. Relationships link two of these objects together in a structured relationship so that they can be associated with an animal. For example a relationship may associate a physical feature and an action to a particular animal that uses the feature to perform the action. The object classes in Creanimate include:

Features. Physical attributes of animals, e.g., beak, claws, fur, small size.

Actions. Activities of animals that are performed with features, e.g., running, swimming, biting, scraping.

Behaviors. High-level, goal-directed activities of animals, e.g., hunting, fleeing predators, attracting a mate.

Animals. Specific animals or abstract categories of animals, e.g., cheetah, raptor, pike, mammal.

Phys-obj's. Physical objects other than animals that are found in the world, e.g., plant, rock, seed.

Properties. Attributes of objects that the system uses for various bookkeeping functions.

Indices. Descriptions of video stories in terms of other objects or relationships.

Impacts. A subjective description of a story or a relationship, e.g., funny, dramatic.

The distinction between actions and behaviors can be confusing. They can be distinguished from each other by their relationships to features and goals. Actions are primitive activities that are directly associated with the features that animals use to perform them. Behaviors are more complex activities that are associated directly with survival goals. In particular, a behavior is an activity that supports only one survival goal, whereas one action may support several behaviors each of which contributes to a different survival goal. Thus, *run*[1] is an action whereas *pursue prey* is a behavior. *Run* is directly associated with the feature *legs*, whereas *pursue prey* is not necessarily associated with any feature. Furthermore, *pursue prey* is associated with a single survival goal, getting food, whereas *run* can serve any number of survival goals, including getting food, avoiding danger, and getting a mate.

Objects are represented as nodes in hierarchical networks with links to objects of both the same and different type. Objects are connected to other objects of the same type in traditional abstraction hierarchies. For example, the actions *walk* and *run* are both connected to *move on the ground* through an abstraction relationship. *Move on the ground* is in turn linked to its abstraction, *move*. Objects are connected to objects of other types in ways that indicate how they relate to each other in the natural world. For example animals have links to the features that they possess and the actions and behaviors that they engage in. Similarly, features are directly linked to the actions that they support, and actions to the behaviors they support. Most links that connect objects to other objects are bi-directional so that the link that connects the feature *teeth* to the action *chew* can be traversed from either *teeth* to *chew* or *chew* to *teeth*. Finally, objects that correspond to animals and attributes in the real world are connected to indices of stories with bi-directional links. Thus, an index contains pointers to the animal and its attributes that appear in the story, and each animal, feature, or behavior contains pointers to the indices for all of the stories in which they appear. These links enable the storyteller to identify stories that are relevant to animals, features, actions, or behaviors by searching through the networks that connect those objects and examining the indices that they are linked to.

Creanimate contains three primary relationships. They connect objects together in the following ways:

Feafuns. A feafun connects a physical feature to an action that it is used to perform, e.g., *long legs in order to run fast*. The name *feafun* is an abbreviation of "feature for a function."

Plans. A plan connects an action to a behavior that it supports, e.g., *run fast in order to pursue prey*.

Bplans. A bplan connects a lower-level behavior to the higher-level behavior that it supports, e.g., *pursue prey in order to hunt*.

Relationships are used to tie features, actions, and behaviors together and associate them with a particular animal. For example, the animal *giraffe* possesses the feafun *long neck in order to reach*. These relationships correspond directly to the explanation questions that the system poses. For instance, the questions "What feature could an animal use to reach for food?" and "Why might an animal have a long neck?" are both answered by the feafun

---

[1] A word about notation: The convention in this and subsequent chapters is that names of concepts in the Creanimate knowledge base appear in italics.

long neck in order to reach. The process by which these relationships and objects are added to the vocabulary proceeds from 1) a taxonomy of the questions that are critical to an understanding of the subject matter to 2) a set of relationships that correspond to those questions to 3) a set of basic concepts that are connected by these relationships.

In addition to the relationships listed above, Creanimate contains a special kind of relationship called a rule.

Rules. A rule encodes a generalization, e.g., "No fish have wings." Rules are known to have exceptions.

Rules differ from the other relationships in Creanimate because the information they encode is not true under all conditions. Rules are used to capture generalizations. However, every rule in Creanimate has at least one significant exception. The role of rules is to trigger expectation-failure remindings about exceptions to those rules.

The objects and relationships described in this section play a critical role in supporting the various example, similarity-based, and expectation-violation reminding strategies.

5.2 Example Remindings

The following transcript appeared in the previous chapter as an example of a suggestion reminding:

```
We need a reason for your dog to fly. Is there a reason you
want your dog to fly?
Student: Give me some suggestions
Canada geese fly. Do you know why canada geese fly?
(This next video is really good. )
Student: Show me a video
[VIDEO: "GEESE MIGRATE"]
```
(Transcript no. Pal-m1-6-5-12.57)

The suggestion reminding strategy is one of three primary example reminding strategies. The other two are the correct answer and the incorrect answer reminding strategies. Each of these retrieves stories that are appropriate for a student's answer to an explanation question. All three rely on the same indexing information in stories and use similar algorithms. The challenge for retrieving example stories is identifying stories that correspond to the particular explanation question just asked and the student's response. This requires information in indices that tell the storyteller which explanation questions apply to a story and an algorithm for searching the story library to find stories for a specific explanation question and response.

5.2.1 Indexing Information for Example Remindings

Since example remindings in Creanimate follow on explanation questions posed to students, the stories in the Creanimate story library are indexed according to the questions that they illustrate. For example, the story about the Canada geese in the example above can answer the following two questions (among others):

Why do animals fly?
How do animals migrate?

These two questions both correspond to the plan *fly in order to migrate.* Therefore, the Canada geese story is indexed under the action *fly,* the behavior *migrate from cold,* and the plan *fly in order to migrate from cold.* In the following fragment of the definition of the index for the Canada geese story, we can see the data structures for these objects in the *behaviors, actions,* and *plans* slots respectively.

```
[index geese-migrate-canadian-geese
     :animals (  [animal canada-geese  ] )
     :behaviors ( [behavior migrate-from-cold  ] )
     :actions ( [action fly  ]... )
     :plans ([plan :action   [action fly  ]
                   :behavior   [behavior migrate-from-cold]])
     ...]2
```

Therefore, the portion of the index that is most important for the example reminding strategy is the portion that indicates the relationships (feafuns, plans, and bplans) that appear in the story. These relationships tell the storyteller which questions and answers the story can be used to illustrate.

5.2.2 The Example Reminding Algorithm

While the reminding algorithm that produces example remindings is simple, it relies heavily on the expressiveness of the indexing vocabulary. The algorithm is implemented as a search through abstraction hierarchies. The starting point of this search and the description of a desirable story are both determined by the current context in the dialogue. As an example, consider the following transcript:

OK, let's find a reason for your frog to have wings. Is there a reason you want your frog to have wings?
Student: So it can fly
That's a good idea. White breasted sea eagles use their feathered wings to help them to fly.
Would you like to see that?
Student: Yes
[VIDEO: "Sea Eagle Catches Fish"] (Transcript no. m2-4-27-14.35)

In this transcript, the explanation question is "Why have wings?" and the student's correct answer is, "so it can fly." Once the dialogue manager has verified that flying is a reason that animals have wings, the correct answer reminding algorithm initiates a search for an example story. This search is guided by the relationship that results from combining the

---

[2] This is an example of the printed representation of the Common Lisp data structure corresponding to a Creanimate object. The text within a set of brackets "[. . .]" refers to a single data structure. The first word always describes the type of data structure, e.g., *animal, feafun, index.* In an object, the second word is the name of the object. The rest of the data structure consists of alternating slot-names (beginning with a colon) and lists of slot values. For more detail, see Chapter 7, The Indexing Vocabulary.

current explanation question and the student's answer. In this case, the current explanation question, "Why have wings?" is an explanation question of the class *Why feature?* A question in Creanimate is represented as a partially filled relationship. For example, the current question is represented as a feafun with the feafun *wings* and a missing action[3]:

```
[feafun :feature [feature wings] :action ?QUERY]
```

This question represents an attempt to identify an absent concept that can fill the specified relationship to the provided concept. In this case, the initial concept is the feature *wings* and the unfilled relationship calls for an action that wings support.

Correct Answer Remindings

When the student in the transcript above gave the answer *to fly*, the storyteller combined them into the following feafun representing *wings in order to fly*:

```
[feafun :feature [feature wings] :action [action fly]]
```

To be an appropriate correct answer reminding, an index must contain this feafun which is called the *target concept*. To find a story that contains the target concept, the Creanimate storyteller searches down abstraction hierarchies starting from one of the values in the target concept. The goal of this search is to find an index that contains the target concept or a specialization of it. Simply stated, the example reminding algorithm is a search that starts with the values provided by the target concept and searches down abstraction hierarchies looking for indices that contain the appropriate combination of these values. The principle behind the example reminding algorithm is:

> A story about the specialization of a concept is an example story about the concept itself.

Therefore, a story about a feafun that is a specialization of *wings in order to fly* is an example story for *wings in order to fly*. To retrieve the example story above, the correct answer reminding strategy starts with the feature *wings* and searches down abstraction links for features associated with stories. Each time it finds a feature that appears in a story, it checks to see if the index for the story contains a feafun that matches the target concept.

The abstraction hierarchy beneath the feature *wings* looks like this:

---

[3] For more details on the representation of questions in Creanimate, see the section on the question data structure in Chapter 6, The Dialogue Manager.

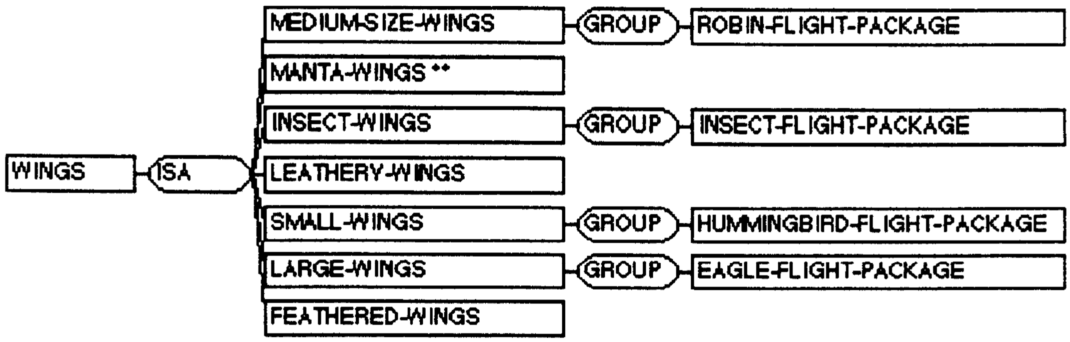

Figure 26. The abstraction hierarchy of features beneath the feature *wings*.[4]

As the reminding algorithm traverses this hierarchy, it looks for indices that match the feafun above. When it reaches *feathered wings,* it finds the index for the sea eagle story that appears in the transcript above. The index for this story contains the following feafun:

```
[feafun :feature [feature feathered-wings]
        :action [action fly]]
```

Because this feafun is a specialization of the target concept *wings in order to fly,* the example reminding algorithm recognizes this story as an example of the current concept and it presents it to the student. Of course, another index that contained a further specialization such as *large wings in order to soar* would also be an appropriate reminding. This reminding is an example of a correct answer reminding. The algorithm for the other two types of example remindings is the same except that the target concept is less tightly constrained, as shown by table 4.

[4] This graph was generated by the indexing tool used to maintain the Creanimate knowledge base. Parent objects (abstractions) appear to the left and children (specializations) to the right. Objects appear in rectangles. Links are labeled with abbreviations of their names that appear in ovals. For more information on the links that connect objects see Chapter 7.

Table 4.–The constraints that apply to correct answer, incorrect answer, and suggestion example remindings. To be selected as an example reminding, an index must contain a relationship that matches the constraints. The constraints in italics are the result of combining the applicable constraints for that type of reminding.

Type of reminding	Constraints	Examples of Constraints
Correct Answer	Current Question	Why wings?
	Student's correct answer	To fly.
	Combined Constraint	*Wings in order to fly*
Incorrect Answer	Category of Current Question	Why wings?
	Student's incorrect answer	To dig holes.
	Combined Constraint	*Some feature in order to dig holes*
Suggest Answer	Current Question	Why wings?
	Combined Constraint	*Wings in order to do some action*

Incorrect Answer Remindings

Like a correct answer reminding, an incorrect answer reminding is constrained by the answer—in this case, unverifiable—given by the student. Consider the following example of an incorrect answer reminding, which appeared in the previous chapter.

```
OK, Let's find a reason for your bear to have a beak.  Is
there a reason you want your bear to have a beak?
Student: So it can smell better
I've never heard of any animals that have beaks to smell, but
there are some other ways that animals smell.
For example, pikes smell.  Pikes use their nostrils to help
them to smell.
I have an interesting video about that.  I like this one
because it has mean fish in it.
Would you like to see a video about that?
Student: yes
[VIDEO: "Pikes smell"]   (Transcript no. m4-5-8-11.37)
```

In this transcript, the student gave the answer, "to smell better" in response to the explanation question, "Why have a beak?" The resulting incorrect answer reminding shows a pike using its nostrils to smell. This reminding is retrieved using a target concept formed by placing the student's answer in an unconstrained relationship of the type used to represent the current explanation question. The target concept for the incorrect answer reminding is generated in the following way:

1. Start with the relationship from the original Explanation Question:

```
[feafun :feature [feature beak] :action ?QUERY]
```

2. Replace the original concept in the explanation question with a wildcard:

```
[feafun :feature [feature feature] :action ?QUERY]
```

3. Replace the unknown concept in the resulting relationship with the student's answer:

```
[feafun :feature [feature feature]
        :action [to smell]]
```

This new target concept can be used to search for incorrect answer remindings. In this case, the storyteller will search the abstraction hierarchy beneath *smell* looking for indices that contain feafuns matching *some feature in order to smell.* In the transcript above, the resulting incorrect reminding shows the feafun *nostrils in order to smell.*

Suggestion Remindings

The target concept for a suggestion reminding is also less tightly constrained than correct answer remindings. Suggestion remindings are only constrained by the current explanation question. The other half of the target concept remains unconstrained. For example, in a dialogue about the explanation question, "How would you like your butterfly to fight?" the question is represented in the following way:

```
[plan :action ?QUERY :behavior [behavior fight]]
```

A story that suggests an answer to that question will contain a plan that shows an action being used to fight. Therefore, the target concept for the suggestion reminding algorithm in a dialogue about *Why fight?* would be:

```
[plan :action [action action] :behavior [behavior fight]]
```

This target concept will match any plan that shows some action being used by an animal in order to fight.

5.3 Similarity-Based Remindings

The similarity-based reminding algorithm presents stories that are similar to but different from an immediately preceding example story. The challenge of creating a similarity reminding algorithm is developing a way for the storyteller to distinguish stories that are similar enough to be interesting and useful from stories that are similar in some respect but do not constitute interesting remindings. The transcript shows an example of a similarity-based reminding.

```
Some shrimp use their transparent bodies to help them hide.
I have an interesting video about that.
Would you like to see that?
Student:    Yes
[Video:  See-through shrimp]
```

That shrimp video reminded me of an interesting video. If you like surprising animals, then you'll love this video. Owl butterflies also have color patterns. Only, instead of being transparent to camouflage themselves, they use their big spots to help them mimic the appearance of another animal.

Would you like to see that? (Transcript no. Pal-m1-6-30-9.37)

In this transcript the student asked for an animal that can hide, and the system engaged him in a discussion of the physical features that animals use to hide. In this example, Creanimate shows the student a story about a variety of shrimp whose transparency enables them to conceal themselves from predators. This story triggered a similarity-based reminding about owl butterflies that have a similar feature, large spots, to serve a similar purpose. These spots scare potential predators away. This reminding helps the student to form generalizations about mimicry and protection as well as providing him with a particularly striking case.

5.3.1 The Original Similarity-Based Reminding Algorithm

Developing an algorithm to produce remindings based on similarity requires an important tradeoff. This tradeoff is between a simpler, more efficient algorithm that requires extra work on the part of the human indexer and a more complex algorithm that reduces the workload of the indexer. Earlier versions of Creanimate used a simpler algorithm that required the indexer to add special information specifically for similarity-based reminding. More recently, this "original algorithm" was replaced with a more sophisticated algorithm that eliminates the need for indexing information that is specific to similarity-based reminding. I present the original algorithm first for two reasons. One, most of the testing with children that is reported in this dissertation was conducted with the original algorithm. Two, it is simpler and therefore easier to present than the improved algorithm. It is easier to describe the most important aspects of the similarity-based reminding algorithm through a discussion of the original algorithm, without the added complexity. Following the description of the original algorithm in this section, I describe the improved algorithm as an incremental enhancement of the original.

Indexing Information for Similarity-Based Reminding: The Abstraction

As I mentioned earlier, the problem for implementing similarity-based reminding is providing a method for the storyteller to reliably determine whether a story is similar enough to another to be an appropriate reminding. The approach used in the original similarity reminding algorithm was to have an indexer make that judgement at the time that he indexes a story. The idea was that the indexer, after indexing the story to indicate how it can be used as an example, would make a judgement as to the range of other stories that would be appropriate similarity-based remindings to follow this story. Of course, one way to do that would be for the indexer to look at all of the possibly similar stories in the story library and determine which of them would be appropriate. Obviously, this solution does not scale-up to large story libraries well, and it would be difficult to maintain lists of similar stories as the story library changes and grows.

Instead, the original similarity-based reminding uses a special representation in each index that the indexer adds for the specific purpose of specifying the range of stories that would be appropriate similarity remindings to follow the current story. As an example, consider the transcript above. The example reminding shows shrimp that use their transparency to camouflage themselves. An indexer might determine that any other story that shows some aspect of an animal's appearance being used to help the animal hide would be an appropriate similarity-based reminding. In this case, the indexer would add a representation that we call an *abstraction* to the index for that story. This abstraction would represent the feafun *appearance in order to hide,* and would communicate to the storyteller that any story that contains an appearance being used to help an animal hide could be a similarity-based reminding to follow this story.

This abstraction is represented in indices in the following way. Recall from the section on example remindings that every index contains representations of the relationships that are depicted in the story. For instance, the shrimp story in the transcript above was shown to the student because its index contains the feafun *transparent color in order to camouflage.* This feafun is represented in the index in the following way:

```
[feafun  :feature [feature transparent-color]
         :action  [action camouflage]
         :abstraction  [feafun :feature  [feature appearance]
                               :action  [action hide]
         ...]
```

We can see that in addition to the feature *transparent color* and the action *camouflage,* this feafun contains an abstraction (underlined). This more abstract feafun, *appearance in order to hide,* plays the role of defining what constitutes a similar story for the purposes of similarity-based reminding. Any story that is encompassed by the abstraction can be considered similar; other stories are not. Each relationship (feafun or plan) that appears in an index is labeled with an appropriate abstraction to support similarity-based reminding.

The Original Similarity-Based Reminding Algorithm

The similarity-based reminding algorithm reverses the example reminding algorithm. As we've already seen, the purpose of the example reminding strategy is to suggest concepts that will resolve the explanation question. In the current example, the feature that was suggested is *transparent color.* The example reminding algorithm starts with the initial concept in the current explanation question and presents new concepts that fill the unfilled relationship in the explanation question.

Example Reminding Algorithm:
Starts with: `[feafun :feature ?QUERY :action [action hide]]`
Result: `[feafun :feature [feature transparent-color] :action [action hide]]`

In contrast, the similarity reminding algorithm starts with the result produced by the example reminding algorithm and works backwards. It looks for other concepts that can take the place of the initial concept in the explanation question.

Similarity-based Reminding Algorithm:
Starts with: `[feafun :feature [feature transparent-color] :action ?QUERY]`

The *similarity* in this algorithm comes from the fact that it does not just use the concept proposed by the example reminding in searching for stories. Using the abstraction as a guide, it also looks at concepts that are similar to the newly-proposed concept. In the current instance, the example reminding strategy started with the action *to hide* and suggested the enabling feature *transparent color.* The subsequent similarity-reminding algorithm starts with the feature *transparent color* and goes in the reverse direction, looking for actions. That is, the example reminding algorithm went from feature to action and the similarity-based reminding algorithm went back from action to feature. This is a general feature of the similarity-based reminding algorithm. When the explanation question is *Why feature?* going from feature to action, the similarity reminding algorithm goes from action to feature. The same applies to explanation questions that go from behavior to action and action to behavior. Thus, the complete similarity-based algorithm has two steps (shown in table 5).

Table 5.–The two step process of the similarity-based reminding algorithm

Similarity-based Reminding Algorithm:

Starts with: `[feafun :feature [feature transparent-color]`
`:action ?QUERY]`

Broadens to similar features:

```
[feafun :feature [feature appearance]
        :action ?QUERY]
```

Result: `[feafun :feature [feature color-pattern]`
`:action [action mimic-another-animal]]`

In this algorithm, the guidance for broadening the search is provided by the abstraction from the index of the previously-told example story. In the transcript above, the example story showed *transparent color in order to camouflage.* Its abstraction was *appearance in order to hide.* Since the example suggested a feature, it is the feature from the abstraction that gets used, i. e., *appearance* for identifying similar stories.

Strong and Weak Similarity Remindings

The original similarity reminding algorithm produces one of two categories of reminding, a *strong* reminding or a *weak* reminding. In a strong reminding, both concepts in the retrieved story fall under the abstraction. In our example that would be a story that showed a type of *appearance* being used for some specialization of the action *hide.* In a weak similarity reminding, only one concept falls under the abstraction. The relevant concept in the abstraction for a weak reminding is the one that corresponds to the new answer to the explanation question. In the current example, the suggested answer was *transparent color,* so the relevant part of the abstraction for a weak reminding would be the abstraction of *transparent color,* which is *appearance.* Thus, to be a weak reminding for the current example only requires that the feature in the reminding be a specialization of *appearance.* A weak reminding would not care about the action in the reminding. In other words, to be a weak reminding for the current example, the feature must be a type of *appearance* but the action need not be a specialization of *hide.* Therefore, the owl butterfly story above is a weak reminding because *large spots* are a type of *appearance,* but *mimicking another animal* is not a way of *hiding.*

Table 6.–The target concepts for weak and strong similarity remindings to follow the "See-Through Shrimp" story

Weak Reminding:

`[feafun :feature [feature appearance] :action [action action]]`

Strong Reminding:

`[feafun :feature [feature appearance] :action [action hide]]`

The Similarity Reminding Search

Simply stated, the similar reminding algorithm starts with the concept proposed by the example reminding and searches for appropriate remindings by examining the concepts that lay close to it in the abstraction hierarchy. The search is limited by the abstraction from the example story. This abstraction can be viewed as an umbrella that delineates a space of similar concepts. This space is searched for concepts that appear in appropriate stories. The search itself is performed using an algorithm that we call *cousin* searching. A cousin search proceeds by making a breadth-first search upwards from the starting point and searching downwards from each parent concept reached in a modified depth-first order. Concepts are only examined for relevant stories during the downward portion of the search, and leaf nodes in the hierarchy are tested before internal nodes. Maintaining a hashtable of visited nodes insures that each node in the hierarchy is only reached once going in each direction.

In the case of the shrimp and owl moth example above, the similarity search would start from *transparent color,* go up its *isa* link to *color* then down the *subtypes* links to the siblings of *transparent color.* (This portion of the hierarchy is displayed in figure 28.) After the depth-first search downward from *color* it would continue its breadth-first search upward to *color-pattern.* From *color-pattern* it would initiate another depth-first search down *subtypes* links. This would lead it to *spots, stripes,* and *camouflage,* as well as their subtypes. At each node on this traversal, the reminding algorithm would check for a story that meets the similarity criteria for the current context. The search terminates either when an appropriate story is found or when all concepts that are specializations of the current abstraction have been searched. Since strong remindings are preferred to weak remindings, the algorithm makes note of the first weak reminding it finds, but it continues to search for a strong reminding until it finds one or the search space bounded by the abstraction is exhausted. If it fails to find a strong reminding, the similarity reminding algorithm falls back on any weak reminding it may have located.

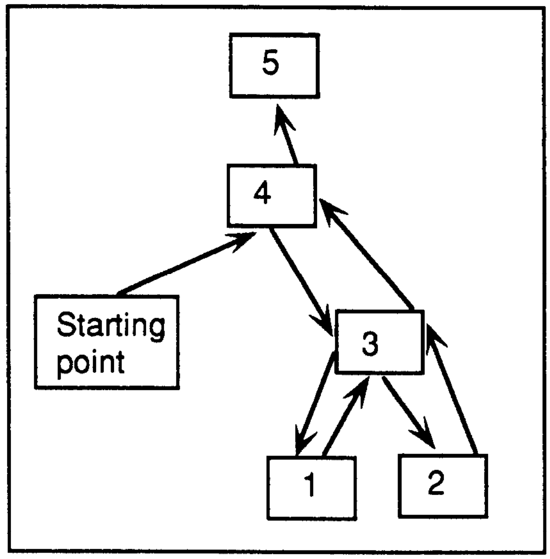

Figure 27. Illustration of the search order in *cousin search*. The traversal of nodes follows the arrows up from child to parent and down from parent to child. The numbers indicate the order in which the nodes get tested. Children are tested before their parents. The search proceeds in breadth-first order in the upward stage from child to parents and in depth-first order in the downward stage from parents to children.

The similarity reminding algorithm is complicated in one additional respect as a result of the goal to present stories that are different in some way from the preceding example story. This is accomplished by filtering out any remindings that have in their index a relationship that matches the current explanation question. In the current example the explanation question is *How to hide?*. Any story that shows a feature being used to hide could be an example story for the current dialogue. Therefore, any story that shows the action *hide* is excluded from similarity-based reminding for the duration of this dialogue. These remindings are reserved for use by the example reminding algorithms since the example reminding strategies are more central to the educational goals of the system. This exclusion leads to the "similar but different" nature of the remindings produced by the similarity-based algorithm. The stories that are retrieved by this algorithm are similar because of the abstraction they share with the example remindings that precede them. However, they are guaranteed to be different in an important respect because they may not show any specialization of the concept that is the basis for the current dialogue (e.g., *hide*). Both the search strategy and the exclusion of example reminding stories are part of both the original and the improved similarity-based reminding algorithms.

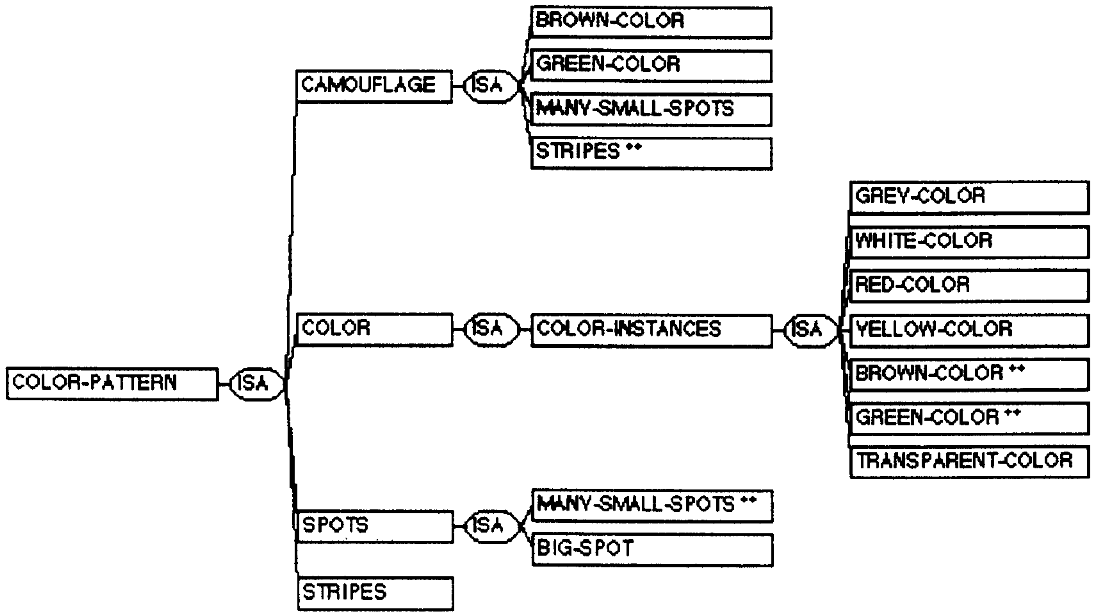

Figure 28. The hierarchy of features beneath the feature *color pattern*

Limitations of the Original Similarity-Based Algorithm

While the original similarity-based reminding algorithm produces valuable remindings, it also has some weaknesses. The first weakness is the need for abstractions. The use of abstractions necessitates work on the part of the indexer that is specific only to the similarity-based reminding algorithm. It requires that the indexer add an abstraction for every relationship in every index. As the knowledge base changes in response to errors and new information abstractions become invalidated and must be updated. As the knowledge base changes, concepts may be moved around in hierarchies so that one concept that had been a parent of another may cease to be so. The bookkeeping involved in constantly modifying abstractions to accommodate changes in the knowledge base is costly. One would prefer that instead of an abstraction being a special annotation to a relationship in an index, that the same knowledge be encoded as a general property of the knowledge base. In other words, rather than placing *appearance* as an abstraction for *transparent color* in the index for the transparent shrimp story, we would like that information to be inferable from general information about the feature *transparent color,* and the action *hiding.*

The second weakness of the original similarity-based reminding algorithm is illustrated by the following transcript:

> Bees dance. Bees use their tails to help them dance.
> I think this video is amazing.
> Is that something you would like to see?
> **Student: Yes**
> *[Video: Bees Dance]*

> That bee video reminded me of an interesting video. If you like mean fish, then you'll love this video.
> Pikes also have tails. Only, instead of having tails to dance, they use their flat tails to help them lunge.
> (Transcript no. pal-m1-6-30-12.40)

In this case, the similarity-based reminding is almost silly. While the particular story that it shows, the pike story, is a dramatic video that few people would regret seeing, the connection between the bee's tail and the pike's is just too tenuous to merit a reminding. It may even cause the student to lose track of the flow of the current dialogue and feel lost after having seen the pike story. The source of the problem in this example is that the reminding hinges on the use of a tail by both bees and pikes; it does not take into account whether or not the uses of those tails are similar. Common sense dictates that possession of a tail is not a sufficient reason to have a reminding based on similarity. The class of animals that have tails is just too broad. However, if the tails in the two stories were more similar or if the actions performed by the animals were more similar, then the reminding would be sensible. The problem is caused by the permissiveness of weak remindings. In this case, *tail* appeared in the abstraction. If the pike story had been a strong reminding, that is, it had contained the entire abstraction, *a tail in order to be visible,* then it would have been a reasonable reminding. However, the storyteller was unable to find a strong reminding in this instance. Instead it found a weak reminding based on the similar feature *tail.* The trouble is that *tail* is too weak a similarity, but for weak remindings the original algorithm is unable to distinguish between *flat tail* which might be a sufficient similarity to merit a weak reminding and *tail* which is not sufficient.

Weak remindings were added to the original similarity reminding algorithm because the strong reminding criterion was too stringent for a system with a limited number of stories. It can be hard to find stories that include both of the concepts in an abstraction but are not excluded because they are examples of the current explanation question. By removing the restriction on one concept in the abstraction we were able to increase the likelihood of finding a similar story that is interestingly different from the current example story. In fact, the good similarity remindings gained by including weak remindings far outweighed the few bad examples like the one above. However, even a rare occurrence of an overly weak remindings can impair a student's overall perception of the system. The problem for implementing weak remindings is that the concepts in an abstraction may be expressed at the appropriate level of abstractness for strong remindings but may be too abstract for weak remindings. When both concepts in the abstraction are used as a target concept, the abstraction may be expressed at the appropriate level of abstractness. However, when only one of the concepts from the abstraction is used, as in weak remindings, that concept may be too abstract. This leads to the problem in the example above. A tail being used to communicate would have been similar enough, but a tail by itself is not a strong enough similarity. In this case, *tail* is at the right level of abstractness when combined with *communicate,* but in the absence of *communicate* it is too general.

5.3.2 The Improved Similarity-Based Reminding Algorithm

The desire to both remove the need for abstractions in indices and eliminate overly weak remindings led to the development of the improved similarity-based reminding algorithm. The key insight that led to this algorithm was the recognition that in Creanimate, relevant similarity is based on both structural and functional similarity, not structural similarity alone. For example, a long neck, a long beak, and a long tail are all similar to each other because they share the quality long. In the context of a story about a long beak being used to reach for things, a story about a long neck being used to reach for things would be relevant, but a story about a long tail being used to swat flies wouldn't. On the other hand, if the first story were about a long beak being used to scratch at biting insects, the story about the long tail would be relevant. Therefore, appearance is not sufficient for making judgements of similarity in Creanimate; similarity also depends on functionality. While several stories may resemble a given story because of the structural similarity of the features involved, the similarity of those features must be relevant to the way those features are used in the stories. In the original algorithm, the storyteller ignored distinctions between structural and functional similarity in its blind adherence to the range of similarity established by the abstraction.

The idea behind the improved similarity-based reminding algorithm is that instead of using an abstraction recorded by an indexer to determine similarity, the storyteller tries to dynamically create abstractions as it examines candidate stories. If it is able to find evidence for an abstraction that links two stories, then it considers them similar. If it doesn't then they are not similar. The actual search algorithm for examining stories is the same as in the original algorithm. The search starts in the same location and proceeds through the same *cousin* search. The difference is in the test for similarity. The original similarity metric checked whether or not the concept in the story it had found was a specialization of the abstraction from the example reminding story. In the improved algorithm, the similarity metric considers whether or not an appropriate context-dependent abstraction can be constructed that covers both the example story and the candidate similar story. An abstraction can only be constructed if there is knowledge in the knowledge base that supports its construction. Looking for knowledge to support the construction of that abstraction is the heart of the improved similarity reminding algorithm.

The improved algorithm works in the following way:

1. Start with the concept suggested by the example reminding.
2. Conduct a cousin search just as in the original algorithm.
3. Try to identify an abstraction that covers both the relationship in the example story and the relationship in the retrieved story.

Consider the following hypothetical example. The storyteller has just presented an example story that shows a squirrel using its teeth to gnaw at a nut. The relevant relationship from this story is:

```
[feafun :feature [feature rodent-teeth] :action [action
gnaw]]
```

In the original similarity reminding algorithm, *rodent teeth* in this feafun would have been labeled with an abstraction that would have likely been *teeth*. The similarity search could have led it to any other type of teeth and shown how they were being used. However, in the improved algorithm, it searches for both a feature similar to *rodent teeth* and an action similar to *gnaw*. Suppose it found a story about a bird using a *beak in order to tear food*. The storyteller would only consider these two stories to be similar if it could find evidence in its knowledge base for an abstraction that encompasses both of these feafuns. The abstraction would have the form below such that *abstract feature* is an abstraction of *rodent teeth* and *beak* and that *abstract action* is an abstraction of both *gnaw* and *bite*:

```
[feafun :feature [feature abstract-feature]
        :action [action abstract-action]]
```

To be acceptable as a similar reminding, this abstraction must meet the following requirements:

1. The features in both the example story and the similar story must be specializations of *abstract feature.*
2. The actions in both the example story and the similar story must be specializations of *abstract action.*
3. *Abstract feature* must "support" *abstract action.*

In order for *abstract feature* to support *abstract action* they must be linked by a *requires, suffices,* or *functionality* link. (These links are described in detail in Chapter 7, The Indexing Vocabulary.) In the case of our hypothetical example, the storyteller would be able to identify the abstract feafun:

```
[feafun :feature [feature mouth] :action [action tear]]
```

Figures 29 and 30 and the following definition of the feature *mouth* show that the beak story is an appropriate similarity-based reminding because 1) *Mouth* is an abstraction of *rodent teeth* through *attachment* and *beak* through *isa;* 2) *Tear* is an abstraction of both *gnaw* and itself; and 3) *mouth* is linked to *bite* through a *functionality* link.

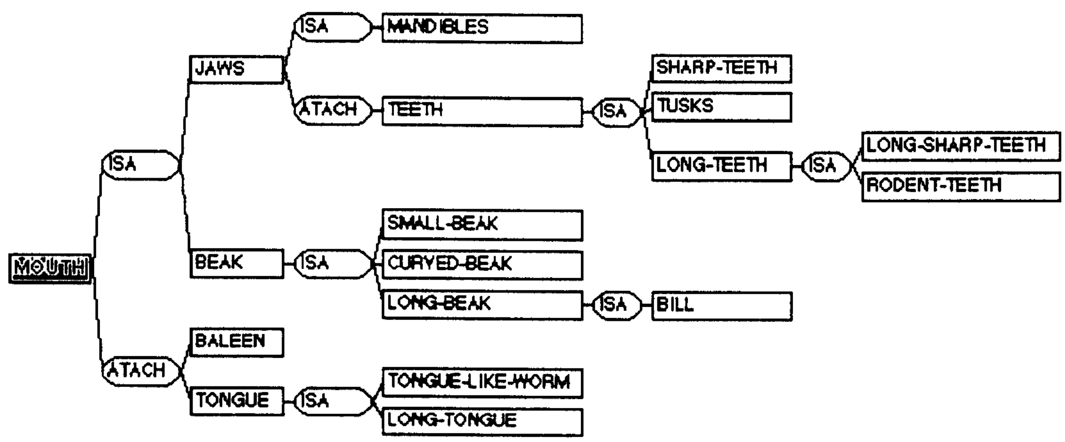

Figure 29. The hierarchy rooted at the feature *mouth*

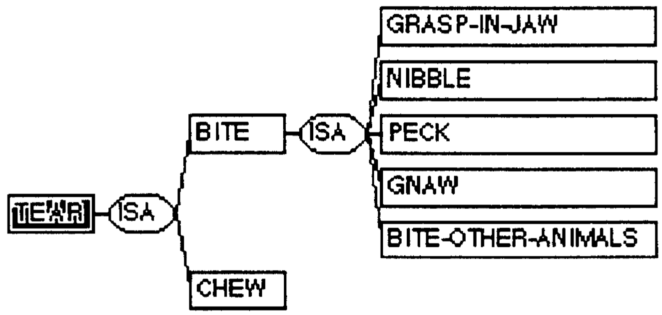

Figure 30. The abstraction hierarchy beneath the action *tear*

```
(DEFFRAME feature mouth
     :subtypes ( [feature beak  ]
                 [feature jaws  ] )
     :actions ( [action carry-prey  ]
                 [action chew  ]
                 [action grasp-in-jaw  ]
                 [action bite-other-animals  ]
                 ...)
     :functionality ( [action probe  ]
                      [action bite  ]
                      [action vocalize  ] )
     ...)
```

This reminding would be introduced in the following way:

> That reminds me of an interesting story. White-breasted sea eagles also use their mouths to tear. Only instead of using their teeth to gnaw, they use their beak to tear.

Besides removing the need to explicitly represent abstractions in indices, the principle advantage of this improved algorithm is that it allows the relationship in which a feature or action is participating in the current story to determine whether or not it is similar to a relationship in a second story. In other words, it allows for context-sensitivity. This means that no longer can two stories be considered similar because of some structural similarity between features *unless* the structural similarity is accompanied by a similarity in the actions supported. Under the old scheme, a *long tail to swat flies* could be considered similar enough to *long legs to run* if the relevant abstraction contained *long appendage.* Under the new scheme, even though they share the abstract feature, *long appendage,* these two relationships would not be similar because the actions do not share an abstraction that is supported by *long-appendage.* However, *long neck to reach for leaves* and a *long tongue to extract termites* would be considered similar because *long neck* and *long tongue* share the parent *long-appendage,* and *long-appendage* supports the action *reach* through a *functionality* link. *Reach* in turn is an abstraction of both *reach for leaves* and *extract termites.*

To summarize, the improved similarity reminding algorithm recognizes that for context-sensitive similarity assessment in Creanimate, similarity of structure is not sufficient. Instead, it relies on similarity of functionality. In addition, it replaces the inflexible and costly pre-specified abstractions with a technique for dynamically generating them in the course of similarity assessment. As of this writing, the improved similarity-based reminding algorithm has only been implemented for similarity of feafuns, not plans or bplans. Therefore, it only appears in discussions of the explanation questions *Why feature?* and *How action?* However, the algorithm applies equally well to the other types of relations.

5.4 Expectation-Violation Remindings

Expectation-violation remindings present stories that are surprising in some way or another. They use this surprise as a source of curiosity, which in turn motivates learning. In order to present surprising stories, the storyteller must have knowledge about the expectations that students have and must know both which stories violate those expectations and how they violate them. This information is represented using rules which encode standard expectations. These rules are associated with the animals, features, actions, and behaviors that they describe, as well as the stories that violate those rules. The three type of rules, only-rules, no-rules, and all-rules each lead to different forms of expectation-violation remindings.

5.4.1 Only-Rule and No-Rule Remindings

The following examples of only-rule and no-rule remindings were introduced in the previous chapter:

<u>Only-rule: Only birds have beaks.</u>

`How would you like to change your butterfly?`

`Student:   A butterfly with a beak.`

`Actually there are animals other than birds that have beaks.`

`For example, octopi have curved beaks.`

`Is that something you would like to see?`

`Student:   Yes.`

`[VIDEO: "OCTOPUS SPAR"]` (Transcript no. m3-5-1-11.33)

<u>No-rule: No fish move through the air.</u>

`So, your pike could have wings to fly.`

`You can commit to that now, or we can look at some other reasons animals have wings and you can decide later.`

`Student:   To fly.`

`Ok, your pike will have wings so that it can fly.`

`[Picture of a pike with wings]`

<u>`Actually there are fish that move through the air.  For example, salmon jump.`</u>

`Do you want to see that?`

`Student:   Yes.`

`[VIDEO:  "Salmon spawn as bear watches"]` (Transcript no. m3-5-7-15.06)

Both of these reminding types respond to a change that the student makes to his animal with a story that contains an expectation-violation. In only-rule and no-rule remindings, the animal in the story violates the same standard expectations that the student's animal does.

Indexing Information for Only-Rule and No-Rule Remindings

In order to support the reminding strategy for only-rule and no-rule remindings, it is necessary for an index to represent the standard expectation that is violated by a story. Every index has a slot for rules which is labeled *expect-fails.* The partial index for the salmon story from the example above shows the no-rule that the story violates:

```
[index salmon-jump-salmon
     :animals ([animal salmon])
     :actions ([action swim-under-water]
               [action jump] )
     ...
     :expect-fails
          ([no-rule :animals ([animal fish  ] )
                    :value [action move-in-air  ]
                    :index [index salmon-jump-salmon]])
     ...]
```

The portion of the index displayed above tells the storyteller that this story depicts a salmon both swimming under water and jumping. It also contains the no-rule which can be read as, "No fish move through the air." The indexer who entered this story into the Creanimate story library felt that this story was surprising because ordinarily people do not expect fish to travel through the air. This no-rule gets activated any time a student changes some fish to make it move through the air in some way. No-rules and only-rules contain a slot labeled *index* that is used by the reminding strategy to identify the particular story that violates the expectation. Only-rules are represented the same way that no-rules are, as this only-rule taken from the index for the octopus story reveals:

```
[only-rule  :animals ([animal bird])
            :value [feature beak  ]
            :index [index octopus-spar  ]
```

This rule provides Creanimate with the standard expectation, "Only birds have beaks." The associated *index* slot indicates that the `octopus-spar` story violates this expectation. This rule gets activated by the only-rule reminding algorithm whenever the student asks for an animal other than a bird with a beak.

Algorithm for Only-Rule and No-Rule Remindings

The algorithms for generating remindings from only-rules and no-rules are virtually identical. They both proceed in the following way. When the student requests a change to an animal, either as a way of starting a dialogue or as a way of resolving an explanation question, any standard expectations that apply to that animal get activated. For an expectation to apply, the only-rule or no-rule must be violated by the student's animal. For example, in the transcript below, the student's bear with wings violates the no-rule, "No mammals have wings."

```
...What would you make?
Student:   A bear with wings.
Actually, there are mammals that have wings.  For example,
fox bats have leathery wings.
Would you like to see that?
Student:   Yes.
[VIDEO: "BATS COOL OFF"]
```
(Transcript no. m3-4-30-15.12)

The no-rule in this example is:

```
([no-rule :animals  ([animal mammal])
          :value  [feature wings]
          :index  [index bats-cool-off-bat]])
```

The only-rule and no-rule reminding strategy is a simple two-step process. First, it searches for a rule that applies to the modification that the student has just made. In this case, that would be the feature *wings*. Second, it determines whether or not the rule applies to the student's animal. In the case of no-rules, the test to determine whether the rule applies checks to see if the animal is a member of the category specified in the rule. In this case, that means the storyteller checks to see if a bear is a mammal. For only-rules, the opposite test applies. An only-rule is relevant if the student's animal is *not* a member of the class specified within the rule. Thus, in the earlier example involving the only-rule, "Only birds have beaks." The test for applicability required that the student's butterfly not be a bird. The search involved in activating relevant rules searches up abstraction hierarchies. To locate the relevant rule for the bear example, the reminding algorithm started with wings and searched up the abstraction hierarchy for any abstractions of wings that had relevant rules. Any rules found in this search receive tentative activation. In the current Creanimate knowledge base, two rules get activated, the one above and the following no-rule that says, "No fish have wings."

```
[no-rule :animals  ([animal fish])
         :value  [feature wings]
         :index  [index manta-ray-feeds]]
```

After completing this first stage, the reminding algorithm looks for the most relevant rule that applies to the student's animal. A rule that applies to a closer ancestor of the student's modification is considered more relevant. In this case, both rules apply to the feature *wings* so neither is more relevant than the other. However, if one had applied to *wings* and another had applied to *support-in-medium*[5], the rule that applied to *wings* would

[5] The feature *support-in-medium* is an abstract feature that captures the similarity of function between wings, flippers, and fins. The natural language equivalent to *support-in-medium* is "appendage to propel through air or water."

be more relevant. So, in the second step, the two rules that were found are examined to see if they apply to the student's animal. Since a bear is a mammal but not a fish, the rule "No mammals have wings" applies, and "No fish have wings" does not. Again, if two rules had applied, the rule with the more specific animal would have been selected because it is more relevant. Like the example reminding strategies, the expectation-violation reminding strategies prefer the story with the highest ranking when there is more than one equally relevant story. Thus, if there had been two stories that both were associated with the rule, "No mammals have wings" the one with the higher rating would have been selected. Once a rule is activated, the appropriate bridge is generated and the student is offered the opportunity to see the story.

To summarize, no-rule and only-rule remindings are triggered by a student's change to his animal. When the student's animal violates the standard expectation expressed by a rule, an expectation-violation reminding is triggered that shows an existing animal that also violates the rule. These remindings surprise students with phenomena that are unfamiliar. This, in turn, increases their interest in and curiosity about the animals they see.

5.4.2 All-Rule Remindings

The following is an example of an all-rule reminding. All-rule remindings differ from no-rule and only-rule remindings in that they are triggered by stories, not by changes to the student's animal. They are also triggered by things that conform to expectations, unlike the other expectation-violation remindings that are triggered by things that violate expectations. All-rule remindings are triggered by stories that depict something that conforms to standard expectations. In the event that the storyteller knows about a story that violates that expectation, it presents an all-rule reminding.

```
Ok, what should we give your frog to help it fly?
STUDENT:  give  it  wings
I like that idea. White breasted sea eagles use their large
wings to help them fly.
I think this video is great.
Do you want to see that ?
STUDENT:  yes
[VIDEO: Sea Eagle Catches Fish]
Birds don't just use wings to fly. For example, black herons
use their wings to make shade.
Would you like to see that?
STUDENT:   yes
[VIDEO: "heron shades water"]
```

In the example above, the sea eagle story conformed to the all-rule, "All birds use their wings to fly." Since the heron story was indexed as a violation of that expectation, the storyteller presented it to the student as an expectation-violation reminding.

Indexing Information for All-Rule Remindings

Just as with the other expectation-violation remindings, All-rule remindings are triggered by default-expectations that are associated with indices. All-rules are slightly more complicated than only-rules and no-rules, however, because they encode not only the expectation but how the expectation is violated by the story. The all-rule that triggered the heron reminding in the previous example looks like this:

```
[all-rule :animals ([animal bird])
          :value [feafun :feature [feature wings]
                         :action [action fly]]
          :expect-viol [feafun :feature [feature wings]
                               :action [action make-shade]]
          :index  [index black-heron-shades-water]]
```

The expectation is recorded in the *value* slot of the all-rule. The expectation in this all-rule translates into English as, "All birds use their wings to fly." In the *expect-viol* slot is found the particular element of this story that violates this expectation. In this case, it is *wings in order to make shade.* (It is unnecessary to indicate which animal violates the expectation because that information can be found in the index from the rule.) Thus, an all-rule contains a standard expectation, a pointer to an index that violates that expectation, and a description of how the story that corresponds to that index violates the expectation.

Reminding Algorithm for All-Rule Remindings

The principle behind the reminding algorithm for all-rules is the same as that for no-rules and only-rules. It follows a two-step process in which it checks rules both for their applicability to the current feafun, plan, or bplan and their applicability to the current animal. In this case, the current animal and relationship are determined by the most recent example, not the student's current animal. The all-rule algorithm starts with an example reminding. After presenting an example story, the reminding algorithm looks for all-rules that apply to the animal in the previous story. This search starts with the animal from the story and proceeds up its abstraction links examining any all-rules found. In the previous example, it started from the animal *white-breasted sea eagle* and traversed *isa* links (displayed in figure 31) until it reached *bird* where it found the all-rule shown above.

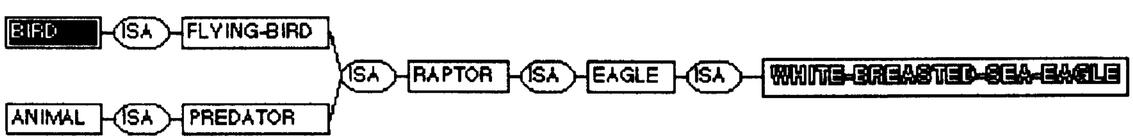

Figure 31. The hierarchy of animals that are abstractions of the animal *white breasted sea eagle*

With each all-rule that the search encounters, the system checks to see if the value in the all-rule applies to the example story that had just been shown. In the transcript above, the white-breasted sea eagle story had been shown as an example of the feafun, *wings in order* *to fly.* This matches the expectation in the all-rule, indicating that this is an appropriate expectation-violation reminding for the current situation. Once again, if several equally relevant stories had been found, the one with the highest rating would have been selected. If the expectation in this rule had not matched the preceding example story, the reminding algorithm would have continued searching up the abstraction hierarchy until it either found an applicable all-rule or it reached the top of the hierarchy. An appropriate bridge to this reminding is constructed by using the *expect-viol* value in the rule and the animal from the index associated with the rule. In this case, the *expect-viol* is *wings in order to make shade* and the animal is *blue heron.* So, the bridge that gets generated is:

```
Birds don't just use wings to fly. For example, black herons
use their wings to make shade.
```

In summary, all-rule remindings are triggered by a stories that match an expectation in an all-rule. They take advantage of predictable stories as an opportunity to show something surprising that will help expand a student's case library of stories and help him to appreciate the diversity of animal behavior.

5.5 Precedence of Reminding Strategies

One of the issues for a system that employs several reminding strategies, any number of which may be active at the same time, is the relative priority of the different strategies. Two such difficulties arise in Creanimate. The first is that one of the incidental reminding strategies may identify a relevant story that would be better used as an example reminding in the same dialogue. The second is that more than one incidental reminding strategy may have a relevant reminding available at the same time, but only one can be presented.

Pending Story Mechanism

The first problem in Creanimate arises when an incidental reminding strategy identifies a story that would also be an appropriate example reminding in the same dialogue. The problem with incidental remindings "stealing" stories that would be more useful as examples has been resolved by instituting a "Pending Story" mechanism. Before initiating a discussion of a new explanation question, the pending story mechanism generates a list of remindings that may be useful as example remindings for that discussion. These stories are marked "pending." Once a story has been marked "pending", incidental reminding strategies are blocked from presenting that story. When a story that is pending gets told, it is no longer marked pending, but is marked "told". A story has its "pending" mark removed when the opportunity to present it as an example has passed. This opportunity passes either with the resolution of the current explanation question or when the relationship the story exemplifies has been discussed without this particular story being shown. Once a story's "pending" mark has been removed, it becomes available for use by incidental reminding strategies.

Once a story has been marked "told" it is treated as if its rating has been lowered beneath the rating of any untold story. Thus, students may see the same story more than once, but the storyteller will not offer a previously told story to a student unless there are no remaining untold stories. The final marking a story can receive is "proposed." A story gets marked "proposed" when it gets offered to the student and the student chose not to see it. A "proposed" story is treated like a "told" story, although previously proposed stories are offered before previously told stories. In other words, when there are several equally relevant stories available, they will be sorted so that untold stories appear first, previously proposed stories second, and told stories third. Within each of those categories, stories will be ordered according to their rating.

In this way, the pending story mechanism blocks incidental remindings from presenting stories that are also good examples for the current discussion. It does so by marking stories that may be presented as example remindings for the current dialogue as "pending." Once a story is pending, it cannot be presented by an incidental reminding strategy. In addition, the same mechanism maintains a record of stories that have been told or proposed to students in order to avoid repeating them.

Priority of Incidental Remindings

The second problem concerning precedence of remindings is that two different incidental reminding strategies may be able to generate an appropriate reminding at the same moment. The problem is deciding which reminding should take precedence when this occurs. This issue arises because incidental remindings are brief digressions from the main discussion of an explanation question. Showing more than one incidental reminding at a time can interrupt the flow of the dialogue too much and may cause the student to become lost. Therefore, only one incidental reminding is permitted for each answer to an explanation question. In the current version of Creanimate, the different incidental reminding strategies have priorities designated in advance such that expectation-violation remindings take precedence over similarity-based remindings.

5.6 Summary

The Creanimate storyteller uses three different reminding strategies, example remindings, similarity-based remindings, and expectation-violation remindings. Each reminding strategy relies on a specific form of information being available in the knowledge base and a specialized search algorithm that enables it to retrieve relevant stories efficiently.

The example reminding algorithms rely on the presence of the relationships that appear in indices. These relationships tell the storyteller the features, actions, and behaviors that are employed in combination in a particular story. The example reminding algorithms use target concepts created from the current explanation question and the student's response in order to identify appropriate examples. The target concept specifies a relationship to be exemplified, and the search algorithm searches through abstraction hierarchies looking for a story that illustrates a specialization of the target concept. The underlying principle is that a specialization of a relationship is an example of that relationship. The search is a breadth-first search down abstraction links that starts from one of the objects that appear in the target concept.

The original similarity-based reminding algorithm relies on the presence of abstractions in indices. Abstractions are generalizations of the specific relationships that are depicted in a story. The abstraction delineates a range of stories that can be considered similar. Essentially, any story that contains a specialization of the abstraction is similar for the purposes of the similarity-based reminding algorithm. However, weak remindings are also permitted, and a weak reminding only needs to match one of the two concepts in the abstraction. The improved algorithm eliminates abstractions and weak remindings and replaces them with a test for dynamically evaluating similarity using general information stored in the knowledge-base not indices. This similarity metric takes into account similarity of function, not just structural similarity. The search for similar stories uses a spreading-activation starting from one of the concepts in the example story and searching "cousins" of that concept in the abstraction hierarchy.

The expectation-violation reminding algorithms use standard expectations which are represented by rules in indices. These all-rules, no-rules, and only-rules encode expectations that are violated by the contents of a story. When a rule becomes relevant to the current dialogue, a reminding is triggered. The search for relevant rules proceeds breadth-first up abstraction hierarchies starting from the current topic of a dialogue.

Chapter 6

The Socratic Dialogue Manager

In this chapter, we move from the storyteller to the task environment. Creanimate's task environment is managed by the dialogue manager, which conducts question-and-answer dialogues with a student. Creanimate's dialogues respond to a student's proposed animal with explanation questions that expose important issues for the adaptability of that animal. While the dialogue manager does not employ all of the techniques that have been associated with Socratic-style teaching (Collins and Stevens 1982; Clancy 1987), it adheres to the Socratic tradition of encouraging a student to make hypotheses and then leading him through an exploration of those hypotheses through thought-provoking questions. The dialogue manager's responses to a student's proposed animal capitalize on the student's investment in his animal to introduce him to the issues involved in animal adaptation. The student's interest in his own animal serves as a natural motivator to engage him in the process of answering the explanation questions that apply to his animal. These explanation questions also provide a context for presenting example stories that supply appropriate answers to these questions.

In this chapter, I describe the structure and implementation of the dialogue manager in Creanimate. Each of Creanimate's dialogues follows a pattern called the *dialogue cycle,* which I describe in Section 6.1. This dialogue cycle forms the basis for the six dialogue plans in Creanimate's current repertoire. These dialogue plans are described in Section 6.2. Finally, in Section 6.3, I describe the underlying implementation of the dialogue plans.

6.1 The Dialogue Cycle

All Creanimate dialogues share an underlying structure. Each dialogue starts with an explanation question and proceeds through the consideration of answers to that question. The pattern that dialogues follow is called the dialogue cycle. However, within the framework of the dialogue cycle, individual dialogues vary a great deal. Dialogues vary for two different reasons. First, the course of a dialogue depends ultimately on the actions and responses of the student. Second, while every dialogue plan includes the basic dialogue cycle, individual dialogue plans contain certain additions to this cycle. These variations are discussed briefly later in this section and in detail in the discussions of the six dialogue plans in the next section.

6.1.1 The Origin of the Dialogue Cycle

The dialogues in Creanimate have their origins in an experiment conducted in 1987. In this experiment seventeen 11 and 12 year old children were told that they were testing an unfinished computer program. However, instead of using a program the children were conversing through a terminal with William Purves, a professor of biology at Harvey Mudd College, who was at another terminal in an adjoining room. The goal of this experiment was to gather valuable information about how children would respond to Creanimate and what they might say. In addition, the conversations conducted by Professor Purves provided an important human model for Creanimate to emulate. The following is a brief example of a dialogue between Professor Purves and an eleven year old girl:

> **Purves:** Okay! I really enjoy talking about kangaroos! Is there something special you would like your kangaroo to do, or to have? Or do you want to ask a question?
> **Student: would want it to have a colorful body**
> **Purves:** That sounds attractive! How might a colorful body be an advantage to the kangaroo?
> **Student: it would be able to blend in to things to hide from danger.**
> **Purves:** Excellent idea! You are talking about colors to hide the animal.
> Many animals have that kind of coloration!
> In a way the kangaroo has that kind of coloration--it is fairly drab, and its surroundings are also fairly drab. Some other animals have very bright colors.
> I am thinking of a couple of examples: the "robin red breast" is a small bird. At certain times of year, the feathers on the chests of the males turn red....(AW, 3/19/87)

An analysis of the transcripts from this experiment revealed that Professor Purves's conversations followed a simple, but effective pattern. This conversational pattern consisted of 1) asking for a modification, 2) asking a question about that modification, and 3) telling a story that related to the student's answer to his question. Using this pattern as a model, we developed the dialogue cycle in Creanimate, which enables the program to produce dialogues that resemble Professor Purves's very closely. (Additional transcripts from these experiments can be found in Appendix E).

6.1.2 The Steps in the Dialogue Cycle

The steps in Creanimate's dialogue cycle are shown below with examples drawn from a student's dialogue.

> **1. Pose Explanation Question**
> Is there a reason you want your pike to have wings?
> **2. Get answer from student or knowledge base.**
> Student: So it can fly
> **3. Present example reminding (if available)**
> That's a good idea. White breasted sea eagles have wings to help them fly.
> This is a great video.

Is that something you would like to see?

4. Allow student to commit to an answer.

So, your pike could have wings to help it to fly. Is that why you would like your pike to have wings?

Student: Show me more reasons

The four steps above are repeated as the student proposes or the dialogue manager suggests different answers. Once the student commits to an answer (step 4), the current dialogue plan terminates, and a new dialogue is initiated. In a real session, the next ordering of the steps listed above can be complicated both by the variations between dialogue plans and by the unpredictable nature of student responses. For example, a student may decide to click on a user control button in order to skip forward, back up, or further modify his animal. In addition, students can give incorrect (unverifiable) answers. As we saw in the previous two chapters, when the student gives an unverifiable answer the storyteller intervenes with an incorrect answer reminding before returning to the main dialogue. This occurs in the transcript below. This excerpt starts with the last step of one dialogue cycle and continues into a second cycle. At the beginning of this transcript, the student commits to swatting as his answer to, "How should we change your dog so it can fight?" The selection of swatting triggers the dialogue initiation process which leads to a discussion of what feature the dog can use to swat.

Step 4. Commit to an answer:

So your dog could swat in order to help it to fight.

How would you like your dog to fight?

Student: By swatting

Ok, your dog will swat so that it can fight...

Step 1. Pose explanation question:

How do you want to change your dog so it can swat?

Step 2. Get answer from student:

Student: Give it something new

Ok, what should we give your dog to help it swat?

Student: Give it wings

Step 3. Example Reminding (incorrect answer):

I don't know of any animals that have wings so they can swat.

But, I do know some other things that wings are used for.

For example, white breasted sea eagles use their wings to help

them keep themselves in the air.

Is that something you would like to see?

Student: Yes

[VIDEO: "Sea Eagle Catches Fish"]

...

Would you like to see more things that have wings, or would you

like to see some ways animals swat?

Student: See ways animals swat

(Step 4 is skipped because of incorrect answer)

<u>Step 1. Pose explanation question:</u>

`Can you think of another way to change your dog so it can swat?`

(Transcript no. m2-4-24-14.28)

The flowchart in figure 32 shows the basic dialogue cycle, as well as the location of possible storyteller interventions for remindings. As the flowchart shows, the nature of a student's response to an explanation question determines the particular path through the dialogue cycle.

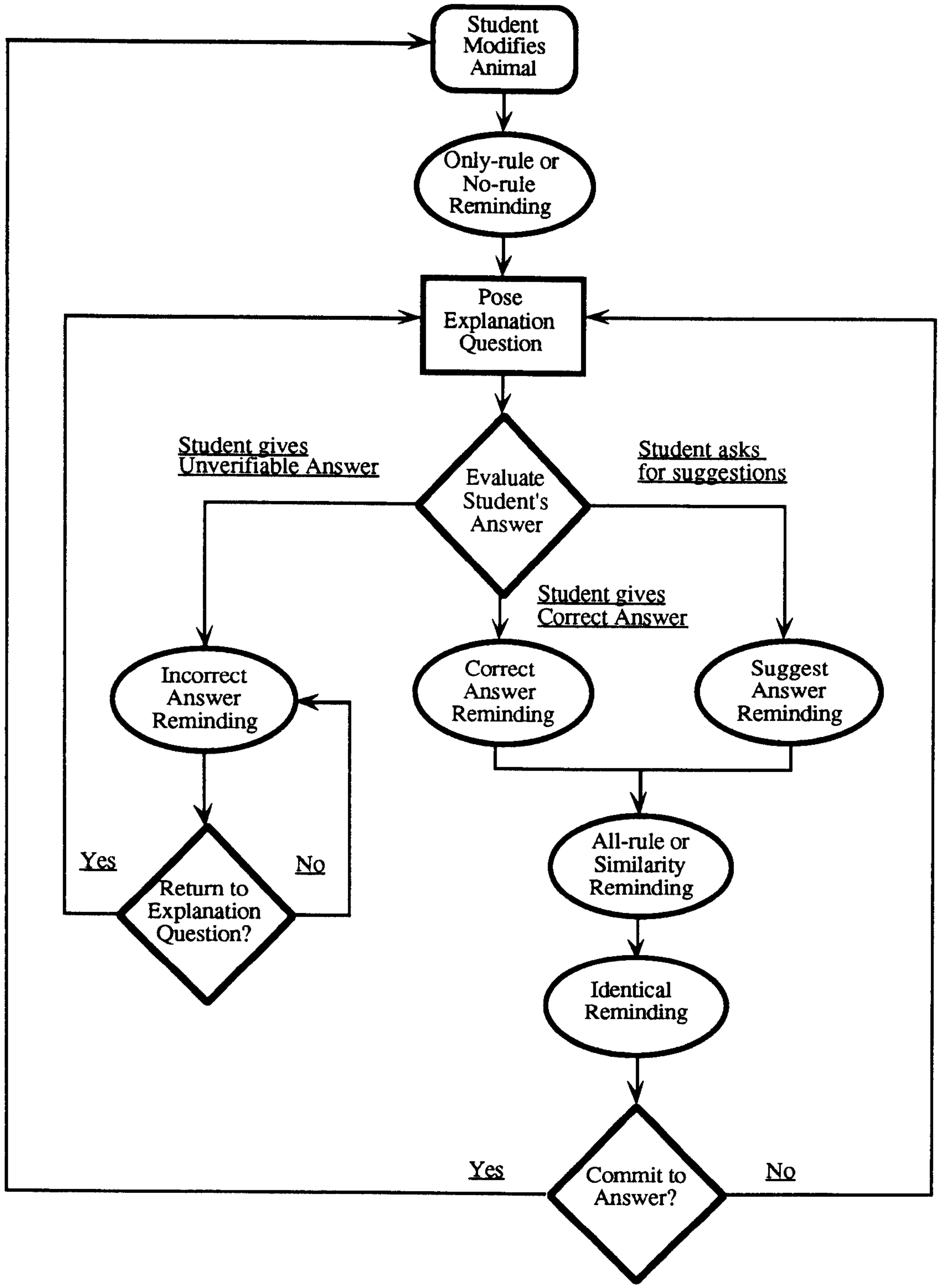

Figure 32. A flowchart of the basic Dialogue Cycle in Creanimate. Opportunities for the storyteller to intervene with remindings appearn in ovals.

6.1.3 Variations from the Basic Dialogue Cycle

All of the dialogue plans in Creanimate adhere to the basic dialogue cycle, but different dialogue plans elaborate on the cycle in different ways. Currently there are two ways in which dialogue plans vary from the dialogue cycle. First, many dialogue plans take advantages of opportunities to examine what the student's animal had or did before the student made a change to it. For example, if a student changes the way an animal gets food, he may get to see how the animal currently gets food in the wild. The step in a dialogue in which the dialogue manager stops to discuss the way an animal actually exists in the wild, is called "considering an animal's current attributes." For example,

```
Before we change your frog's legs we should look at what its
old legs were good for.  Do you know what frogs use their
strong hind legs for?
(I have a cool video about that.  This video also has snakes
in it.)
```
(Transcript no. m1-5-8-13.59)

Second, several dialogue plans contain extensions that enable the dialogue manager to continue a discussion even if it runs out of answers for a particular explanation question. In these dialogues, if the student has seen all of the answers for a question that exist in the Creanimate knowledge base, the dialogue manager will extend the dialogue by considering answers to similar questions. For instance, if the dialogue manager runs out of reasons for an animal to run fast, it can extend the discussion by suggesting some reasons that animals fly or swim fast. In the following transcript, the student has exhausted the uses that Creanimate knows about for beaks.

```
Well, that's the only video I have about why animals have
beaks, but I know about some things that are like beaks.
Maybe one of them will give you a reason you'll like for your
frog to have a beak.
Mandibles are like beaks, because they are both mouths.
Locusts have mandibles. Locusts have mandibles to help them
to saw.
I have a cool video about that. This is a close up video.
Do you want to see that ?
```

Both considering an animal's current attributes and extending dialogues to consider similar attributes are discussed in more detail in the discussion of the dialogue plans that include these variations from the dialogue cycle.

6.1.4 Scriptedness Versus Disorientation in the Dialogue Cycle

Following a simple plan like the dialogue cycle can be both a strength and a weakness for an educational system. The consistent structure of the dialogue cycle is a strength because it helps the student to stay oriented throughout an extended interaction. Because the discussion is open-ended and is often interrupted for stories that run from 20 seconds to a minute, there is a constant risk that a student will lose track of where he is in a dialogue or that he will forget what the current topic is. The simple structure of the dialogue is designed to minimize the likelihood of the student becoming lost. The consistency of the dialogue cycle reduces the chance of the student becoming lost, and it provides him with frequent reminders of where he is in his discussion. In addition, two of the user control buttons, "What's the point?" and "Big Picture" give the student immediate access to orienting information.

However, the same structure that helps the student to keep track of the discussion also raises the risk of the dialogue seeming excessively scripted. Four different features of the Creanimate dialogue manager enable it to avoid a scripted feel while still maintaining the consistency of the dialogue cycle. First, every dialogue starts with an open-ended question. The fact that the student is free to answer in an unconstrained fashion and the program responds in different ways to different answer gives the dialogue a sense of openness not scriptedness. Since the student provides the initiative for the dialogue not the computer, the dialogue has an unconstrained feel. Second, dialogue plans each elaborate upon the dialogue cycle in significant ways. While they all have at their heart the same dialogue cycle, they each contain variations. Third, the output generated by the system continually varies. At the same point in the dialogue cycle on different passes through the same dialogue plan, the language used by the program will differ. For example, in the *Why-action?* dialogue, the same step (Step 4, the commitment point) may appear the following way on two successive passes:

```
So, your frog might fly to pursue its prey.
You can commit to that now, or we can look at some other
reasons animals fly and you can decide later.

So, an animal could fly to pursue its prey.  Is that what you
want for your frog?
```

Varying the program's output this way makes the dialogue seem more natural. It prevents the student from feeling that the dialogue is overly scripted while still giving him enough information to keep track of where he is. For example, in the text above, the student is able to recognize in both cases that he is at a commitment point, however, it avoids the mind-numbing regularity that one might fear from a computer following a simple cycle.

Fourth, depending upon how the student responds to the explanation question on each pass through the cycle and what stories are available for incidental remindings, a single dialogue will differ greatly from pass to pass. Specifically, the same dialogue may get interrupted for different kinds of expectation-violation or similarity remindings. Thus, even though a student may make several passes through the same dialogue cycle in the course of considering the question, "Why should a dog have a big nose?" each time he might see different sorts of example remindings introduced in different ways and accompanied by different sorts of incidental remindings. For example, the first time through the dialogue cycle, the student might give a correct answer and see a correct answer reminding followed by an expectation-violation reminding and two identical remindings; the second time he might give an unverifiable answer and see an incorrect answer reminding; the third time he might ask for suggestions and see a suggestion reminding introduced by a question and followed by a similarity-based reminding.

So while all the dialogue plans in Creanimate have at their heart the same dialogue cycle, the variety in student responses, the variety among dialogues, the variety in the language generated by the dialogue manager, and the variety in the types of remindings prevent these dialogues from feeling overly scripted. In general, the open-ended questions and the different reminding strategies provide a sense of freedom and variety whereas the consistency of the underlying dialogue provides structure. The dialogue cycle provides a counter-balance to the potentially disorienting feel of a discussion punctuated by stories. This structure helps the student to maintain a sense of orientation through a free, open-ended interaction.

6.2 The Creanimate Dialogue Plans

There are six dialogue plans in the current version of Creanimate. They are:

- Why-feature-dialogue
- How-action-dialogue
- Check-action-dialogue
- Why-action-dialogue
- How-behavior-dialogue
- Why-behavior-dialogue

Each dialogue plan conducts a discussion of a particular explanation question. There is one plan for each category of explanation question except for *How action?* which has two plans, *how-action-dialogue* and *check-action-dialogue*. In this chapter, I describe each dialogue plan in detail including the conditions under which the dialogue plan is initiated, the answers that the dialogue manager will accept for its explanation question, and the ways in which the dialogue plan elaborates on the basic dialogue cycle.

6.2.1 Why Feature Dialogue

The *why-feature-dialogue* is triggered when a student asks to add a physical feature to his or her animal. The goal of this dialogue is to find an action that is supported by the chosen feature. In other words, it tries to find the action to fill in a feafun that includes the student's requested feature. If the student responds to the question with an action, the dialogue manager evaluates the answer to see if it can find an animal that uses the current feature to help it perform the action. If it does, the storyteller will attempt to use the resulting feafun as the basis for a correct answer reminding. If the dialogue manager cannot find an animal with the given feafun, then the storyteller will attempt an incorrect answer reminding. The following transcript shows one pass through the *why-feature-dialogue* by a student:

<u>Proposing a new animal</u>

```
Suppose you could create a new animal by taking an existing
animal and changing it some way.  What would you make?
```
`Student: A mouse with wings.`
```
Unfortunately, I don't have any pictures of a mouse, so you
will not be able to see the animal you make. We can still
talk about it though.
```
`Student: Go on.`

<u>Expectation violation reminding:</u>

```
Actually there are mammals that have wings. For example, fox
bats have leathery wings.
```
`Do you want to see that ?`

`Student: Yes`

`[VIDEO: "BATS COOL OFF"]`

Explanation question:

We should come up with a reason for your mouse to have wings.
Why would you like your mouse to have wings?

Student: So it can so it can [sic[1]] **fly**

Correct answer reminding:

That's a good idea. White breasted sea eagles use their feathered wings to help them fly.
Would you like to see that?

Student: Yes

[VIDEO: "Sea Eagle Catches Fish"]

Identical reminding:

White breasted sea eagles are not the only animals that use their wings in order to fly. For example, black skimmers have wings to help them to fly.

I have a great video about that.
Would you like to see a video about that?

Student: What other videos can I see?

I can show you videos about all of these animals that have wings. Which would you like to see first?

black skimmer
Swainson's hawk
robin
falcon
goose

Student: falcon

Falcons have wings to help them to fly.
I have a pretty good video about that. This has high speed chases in it.
Is that something you would like to see?

Student: Yes

[VIDEO: "FALCON AND QUAIL"]

...

Commitment point:

So, your mouse could have wings to fly.
You can commit to that now, or we can look at some other reasons animals have wings and you can decide later.

Student: To fly

Ok, your mouse will have wings so that it can fly.
(Transcript no. m3-5-15.~34)

[1] Student re-typed "so it can" unnecessarily.

In this example, the student answered the question, "Why have wings?" with the action *fly*. Since the goal of the dialogue is to find an action to go with the feature *wings*, this is an appropriate answer. But what would happen if the student answered with a behavior instead? While Creanimate's knowledge base is divided into features, actions, and behaviors, a student's understanding may not necessarily follow the same distinctions. When a student gives a behavior as an answer to the question *Why feature?* the evaluation of the student's answer is more complicated. To verify that a behavior is a correct answer, the dialogue manager must find an action that is supported by the feature and that in turn supports the behavior given by the student. For example, suppose a student asked for a fish with wings, and then gave the reason "so it can chase birds," which the parser interprets as the behavior *pursue prey*. To accept that as a correct answer, the dialogue manager must find an intermediate action that is connected with *wings* and *pursue prey*. This action would appear somewhere in both a feafun with *wings* and a plan with *pursue prey*. The following transcript depicts this scenario:

```
If your pike is going to have wings, that should help it to
do something. Why would you like your pike to have wings?
Student: So it can catch birds
That might work. Black skimmers use their wings to fly.
Flying helps them to pursue their prey.
This is a great video.
Would you like to see that?
```

The *why-feature-dialogue* diverges from the standard dialogue cycle in two ways. First, once the student commits to a reason for having the feature, the dialogue manager stops to consider any features that the animal currently has that will be replaced by the new feature. Second, if the student does not choose any of the actions that Creanimate knows can be supported by the feature, it will extend the dialogue by considering reasons for having features that are similar to the student's selected feature.

Considering the Animal's Current Feature

The portion of the *why-feature-dialogue* that considers an animal's current attributes checks to see what the new feature replaces. For example, in the following transcript, a student asked for a bear with a beak. After settling on a way for the bear to use its beak, the *why-feature-dialogue* stopped to consider how bears use their current beaks.

```
Ok, your bear will have a beak so that it can dig.
Student: Go on.
Before we change your bear's mouth we should look at what its
old mouth was good for. Bears have mouths to help them to
bite.
I think this video is great.
Is that something you would like to see?
Student: Yes.
[VIDEO: "BEAR EATS"]
```

(Transcript no. Pal-m2-6-25-10.44)

In order to show this, the dialogue manager must be able to identify features that are essentially replacements for each other. This is done by associating the property *natural kind* with certain features. When a feature has the property *natural kind* it means that an animal can possess only one specialization of that feature. Thus, if the student adds a new feature to his animal and the dialogue manager ascertains that the animal already has another feature of the same *natural kind*, then the dialogue manager assumes that the new feature replaces the old feature. It then asks the student to consider what the old version of the feature was used for. In the transcript above, the feature *mouth* has the property *natural kind* and the feature *beak* is represented as a type of *mouth*. Thus, when the student added a beak to his bear, the storyteller assumes that the beak must replace the bear's current mouth.

Extending the Dialogue by Considering Similar Features

The *why-feature-dialogue* also diverges from the standard dialogue cycle whenever the dialogue manager exhausts all the reasons it knows for having a particular feature. If the dialogue manager reaches this point without the student committing to an answer then it launches a variant of the *why-feature-dialogue* known as the *similar-why-feature-dialogue*. The *similar-why-feature-dialogue* attempts to answer the current explanation question by proposing reasons that animals use similar features. The algorithm for identifying similar features resembles the one used by the improved similarity-based reminding algorithm. It looks for other features that are linked to the original feature by parents that share not just a structural similarity but a functional similarity. For example, in the following transcript, Creanimate exhausts the reasons it knows for having a beak without the user selecting one of them:

```
Well, that's the only video I have about why animals have
beaks, but I know about some things that are like beaks.
Maybe one of them will give you a reason you'll like for your
frog to have a beak.
Mandibles are like beaks, because they are both mouths.
Locusts have mandibles. Locusts have mandibles to help them
to saw.
I have a cool video about that. This is a close up video. Do
you want to see that ?
```

In this way, the dialogue manager is able to extend the why-feature-dialogue by proposing reasons for having similar features.

6.2.2 How Action Dialogue

The *how-action-dialogue* is the inverse of the *why-feature-dialogue*. Like the *why-feature-dialogue*, it tries to complete a feafun, but instead of looking for an action to go with a student's chosen feature, the *how-action-dialogue* looks for a feature to support the student's chosen action. The *how-action-dialogue* is initiated in one of two circumstances. The first occurs when a student directly requests an action for his animal as the initial modification. The second occurs when a student resolves a *how-behavior-dialogue* by selecting an action to support a behavior. In both cases, the dialogue manager launches the *how-action-dialogue* because it recognizes that the student's animal has an action without an accompanying feature.

Thus, the objective of the *how-action-dialogue* is to identify a feature that supports the action in some way. The *how-action-dialogue* is not concerned with whether or not the feature is sufficient on its own to enable the animal to perform the action. That concern is the responsibility of the *check-action-dialogue* which will be discussed next. The test for whether or not a feature is considered an acceptable answer to the explanation question is simply whether or not the feature appears with the action in a feafun of some animal in the Creanimate knowledge base. As the transcript below shows, the answer options associated with the *how-animal-does-action* question data structure prevent the student from giving a response other than a feature. However, the student is free to have the animal perform the action by using an existing feature of his animal, changing an existing feature, or giving it an entirely new feature.

```
How do you want to change your butterfly so it can hide?
   Change its ...
   Give it something new.
   Use its current ...
   Give me some suggestions.
Student: Give it something new.
Ok, what should we give your butterfly to help it hide?
   Give it ...
   Make it ...
   Give me some suggestions
Student:   Make it very small.
```

It is always possible for a student to give an unverifiable answer like the following, however:

```
Can you think of another way to change your dog so it can
fly?
Student: Use its current front legs
I don't know exactly what type of forelegs dogs have...
I don't know of any animals that have forelegs so they can
fly.  But, I do know some other things that forelegs are used
for.
For example, bears use their forelegs to help them wrestle.
Do you want to see that?        (Transcript no. m4-5-8-15.24)
```

An unverifiable answer for the *how-action-dialogue* is one for which the dialogue manager is unable to find any examples. In the transcript above, it was unable to find an animal with the feafun *forelegs in order to fly*. Unverifiable answers trigger incorrect answer remindings.

The *how-action-dialogue* also includes a step to consider the animal's current attributes. In particular, it looks at the original animal's features. Once the student has committed to a feature that answers the explanation question, the dialogue manager examines the animal to see if it currently has some version of that feature. If it does, the dialogue manager will show the student how the animal's current feature is used, just as in the *why-feature-dialogue*:

So your frog could use legs like gulls have to help it dance.
You can commit to that now, or we can look at some other ways animals dance and you can decide later.
What would you like to give it?

Student: legs

Before we change your frog's legs we should look at what its old legs were good for. Do you know what frogs use their strong hind legs for?

(I have a cool video about that. This video also has snakes in it.)

Student: So they can high jumps

Yes, that is right...*[Buggy output deleted]*...frogs have strong hind legs to help them to leap.
Would you like to see a video about that?

Student: Yes.

[VIDEO: "FROG AND SNAKE"] (Transcript no. m1-5-8-13.59)

At the conclusion of the *how-action-dialogue* the student commits to a feature that the animal will use to perform the new action. This feature can either be a new one or one that the animal already has. Verifying that this feature is sufficient to accomplish the action is the responsibility of the *check-action-dialogue.*

6.2.3 Check Action Dialogue

The *check-action-dialogue* is initiated whenever a student completes a feafun by providing either a feature or an action. This occurs at the completion of either a *why-feature-dialogue* or a *how-action-dialogue.* The explanation question for the *check-action-dialogue* is the same as that for the *how-action-dialogue,* "How should we change your animal so that it can *action?"* However, the role of the *check-action-dialogue* is to look at the requirements of the action to make sure that the animal has the complete set of features necessary for the new action. If the animal is found lacking in some way, the dialogue manager will ask the student to make additional modifications.

The dialogue manager looks at three aspects of the action in assessing whether the animal needs further modification. It looks at an action's parts, the features it requires, and the features that suffice for the action.[2] The parts of an action are other, more primitive actions. For example, the action *fly* is composed of *take off, generate lift, steer in flight,* and *land.* If an action has parts, the *check-action-dialogue* looks at each of the parts and checks to see whether the animal is adequately outfitted to perform them. An action may require several individual features all of which are necessary for an animal to perform the action. If an action requires more than one feature, then the *check-action-dialogue* will look to see if the animal has all of them present in some form. Otherwise the action will have a list of sufficient features, any one of which is sufficient for an animal to perform the action. As an

---

[2] These are implemented through the slots of an action frame named *parts, requires,* and *suffices.* They are discussed in Chapter 7, The Indexing Vocabulary.

added complication, any of the required or sufficient features may be implemented as a group of features. If this is true, then the *check-action-dialogue* must ensure that every feature in that group is present in order for the animal to perform the action.

To see how the *check-action-dialogue* works, let's look at the way it handles flying. As I mentioned before, the action *fly* has several parts. It also requires *wings*. These are shown in the portion of the object definition for *fly* that appears below:

```
(defframe action fly
     :isa ([action move-long-distance] [action move-fast]
           [action move-in-air] [action move-above-ground])
     :subtypes ([action soar])
     :parts ([action generate-lift] [action steer-in-flight]
             [action flight-takeoff] [action flight-landing])
     :features ([feature insect-wings]
                [feature medium-size-wings]
                [feature small-wings])
     :requires ([feature wings] )
     ...)
```

For this example, we will assume that the student started by asking for a dog with wings. This triggers a *why-feature-dialogue* in which the student commits to flying. In response, the dialogue manager initiates the *check-action-dialogue* to identify any additional changes that need to be made to the student's animal to enable it to fly. It is obvious to us that a dog is too large to fly, so its size must be reduced. However, the process by which the dialogue manager recognize that fact is complicated. The first step of the *check-action-dialogue* plan is to look at the actions requirements. In the case of *fly* the only value in the *requires* slot is *wings*. The dialogue manager sees that the student has already given the dog the feafun *wings in order to fly* so this requirement has been fulfilled. The next step is to look at the parts of flying. The action *fly* has several actions in its *parts* slot which the dialogue manager must look at one by one. The first one is the action *generate lift* . (The program outputs the action *generate lift* in natural language as "keep itself up in the air.") A portion of the definition for *generate lift* appears below:

```
(defframe action generate-lift
     :isa (      [action move-in-air  ] )
     :part-of (  [action fly  ] )
     :suffices ( [feature robin-flight-package  ]
                 [feature hummingbird-flight-package  ]
                 [feature insect-flight-package  ]
                 [feature eagle-flight-package  ] )
     ...)
```

When the dialogue manager looks at *generate lift*, it recursively initiates the *check-action-dialogue*. In this case, however, the action has no values in its*requires* or *parts* slot, so the dialogue manager looks at the *suffices* slot. In the *suffices* slot, there are several features that all contain the string "flight-package" in their names. Each of these flight-packages is really a feature group that includes a body-size and a type of wing. For example, *eagle-flight-package* below consists of a medium-sized body and large wings. This means that one way for an animal to "keep itself up in the air" is to have the combination *medium size* and *large wings*.

```
(defframe feature eagle-flight-package
     :isa ([feature flight-package])
     :group ([feature large-wings]
             [feature medium-size])
     :actions ([action generate-lift]
               [action soar])
     :properties ([property vanilla-group])
     :suffices-for ([action generate-lift])
     ...)
```

Groups provide a primitive but effective way for indicating the different ways features can combine to support an action. The challenge of implementing the conditions for *generate lift* in Creanimate is representing the fact that a certain ratio of wing-size to body weight must be maintained for an animal to stay aloft. The way this problem is solved is to introduce groups of features, each of which is sufficient for generating lift. In order to generate lift, an animal must have all the features that comprise one of these groups. The groups that are sufficient for *generate-lift* are:

Group	Elements
eagle-flight-package:	medium size, large wings
robin-flight-package:	small size, medium sized wings.
hummingbird-flight-package:	very-small-size, small wings
insect-flight-package:	very small size, insect wings

The fact that each of these packages appears in the *suffices* slot for *generate lift* tells the dialogue manager that any one of them is sufficient for an animal to stay aloft. However, an animal must have each of the features that makes up the group. Since, a dog has the feature *medium size* the dialogue manager recognizes that it is closer to achieving the *eagle flight package* than any of the other. Therefore, it will suggest to the student (if he asks for suggestions) that he give his dog *large wings* in place of the ordinary *wings* that it currently has. However, if the student proposes on his own accord to make the animal smaller, the dialogue manager will help the student to give his animal the appropriate flight package for his new, smaller dog.

Once the sufficient conditions for *generate lift* are met, the dialogue manager will go on to consider the conditions necessary for achieving the other parts of the action *fly*, such as giving it a tail that will enable it to steer while it is flying. The abbreviated transcript below shows a student engaged in a *check-action-dialog* for an animal that can fly.

The first part of flying, steering:

```
In order to fly, animals must be able to steer while they are
flying. Let's talk about how your dog will steer while it is
flying.
Having wings is not enough for your dog to steer while it is
flying. It needs more.
```

```
How else do you want to change your dog so it can steer while
it is flying?
```
. . . *(Student adds a feathered tail.)* ...
```
OK, your dog will use its feathered tail to help it steer
while
it is flying.
```
<u>The second part of flying, keeping itself in the air (*generate lift*):</u>
```
Flying also requires an animal to be able to keep itself up
in the air. We should figure out how your dog will keep
itself up in the air.
Having wings is not enough for your dog to keep itself up in
the air. It needs more.
How else do you want to change your dog so it can keep itself
up in the air?
```
Student: Give it light bones
```
That's a good idea. Dogs could use hollow bones to help them
to keep themselves in the air.However, I don't have any video
that shows this.
```
. . . *(Student gives it hollow bones.)*

<u>*Hollow bones* support *generate lift* but are not sufficient:</u>
```
Having hollow bones is not enough for your dog to keep itself
up in the air. It needs more.
How else do you want to change your dog so it can keep itself
up in the air?
```
Student: Give me some suggestions

. . . *(Student sees suggestion remindings for several different groups of features.)*

<u>A suggestion reminding for *medium size* and *large wings* (*eagle flight package*):</u>
```
Medium size and large wings can work together to help an
animal to keep itself up in the air.
Canada geese use their medium size and large wings to help
them keep themselves in the air.
This is a good video.
Would you like to see that?
...
```
<u>Commitment point for *generate lift*:</u>
```
What would you like to add to your dog so it can keep itself
up in the air?
```
Student: Medium size and large wings
```
OK, your dog will use its medium size and large wings to help
it keep itself up in the air.
```
(Transcript no. Pal-m2-6-30-9.56)

To summarize, the order of steps in the *check-action-dialogue* is:

1. Check if the action are any required features. If so, make sure that the animal possesses them all.

2. Check if the action has parts. If so, make sure the animal possesses all of the features necessary to support those actions. (Recursively initiate the *check-action-dialogue* with each part.)
3. If the action has no requirements or parts, make sure that the animal possesses at least one of the sufficient features.

For an animal to *possess* a feature, the animal must have the feature or a specialization of the feature. If the feature is a group, the animal must have all of the parts of the group. For an action that is a part of the current action to be *supported*, it must itself pass all of the tests for the *check-action-dialogue*. The *check-action-dialogue* plan does not have any extensions for when it runs out of answers to suggest, however, it does stop to consider the student's initial animal in the same way that the *how-action-dialogue* does. Once the student has committed to a new feature, it looks back at any similar feature that the animal might have already had, and discusses the animal's uses for that feature.

6.2.4 Why Action Dialogue

The *why-action-dialogue* seeks a behavior to complete a plan. It is triggered either when a student adds an action to his animal as an initial modification, or when he adds one as a reason for a new feature. An acceptable answer to the *Why action?* explanation question is any behavior that the action supports either directly or indirectly. To evaluate the correctness of a behavior suggested by the student as an answer to this question, the dialogue manager checks the knowledge base for the existence of a plan that either includes both the action and the behavior or one that includes the action and another behavior that supports the behavior provided by the student. So, if the student has suggested hunting as a reason for an animal to hide, the dialogue manager would look for either the plan *hide in order to hunt* or a plan involving *hide* and some other behavior that supports hunting in some way. Thus, the dialogue manager would accept *hunt* because the knowledge base contains animals with the plan *hide in order to avoid prey detection.* The behavior *avoid prey detection* assists *get close to prey* which in turn is part of the behavior *hunt* as shown in figure 33.

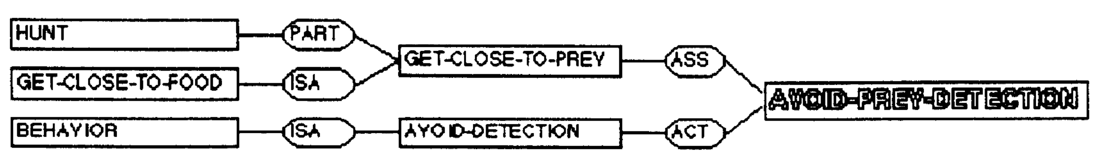

Figure 33. The abstractions of the behavior *avoid prey detection.* The hierarchy displays parents to the left of children so that *avoid prey detection* is connected to *get close to prey* by an *assists* link. *Get close to prey* is in turn connected to *hunt* by a *part-of* link.

If the dialogue manager is unable to find a connection between the action and the behavior suggested by the student, then the storyteller will attempt to present an incorrect answer reminding. If, instead of proposing an answer, the student asks for suggestions, the dialogue manager identifies correct answers by identifying plans that the action participates in and suggesting the behaviors that appear in those plans.

The *why-action-dialogue* elaborates on the standard dialogue cycle with both a consideration of the animal's current activities and an extension that considers reasons for similar actions. It stops to consider the animal's current actions or behaviors after each correct answer considered (regardless of whether the answer was suggested by the student or the dialogue manager). For each behavior discussed as a reason for the student's action, the dialogue manager considers how the animal currently achieves the same objective. For example, in the following transcript, the student asked for a bear that can dance. After seeing bees dance to tell other bees where to find food, the dialogue manager helps the student to consider how butterflies get their food.

Pose the explanation question:

We need a reason for your butterfly to dance. Is there a reason you want your butterfly to dance?

Student: Give me some suggestions

...

Suggestion reminding:

We've already seen that bees dance. Do you know why bees dance?

Student: Show me a video

[VIDEO: Bees dancing]

Now, do you know why bees dance?

Student: Tell me

Similarity-based reminding:

Hey, that bee reminded me of an interesting video. This has lions in it.

Hyenas also communicate. Only, instead of dancing to tell other animals where to find food, they bark to warn their friends about danger.

Is that something you would like to see?

...

Consideration of current animal's behavior:

Bees tell other animals where to find food. Your butterfly could dance to tell other animals where to find food. That might help it to find food like bees do. Do you know how butterflies currently find food?

Student: I don't know

Butterflies probe to locate nectar. That is how they find food.

Commitment point:

So, a butterfly might dance to tell other animals where to find food. Why would you like your butterfly to dance?

Student: Show me more reasons (Transcript no. m2-4-27-12.36)

In addition to the consideration of an animal's current behaviors shown above, the *why-action-dialogue* also varies from the standard dialogue cycle if it exhausts all of the behaviors it has for a particular action. Just as with the *why-feature-dialogue*, if it runs out of reasons for an action, it will start to suggest reasons for similar actions. It locates these actions by searching for actions that share an abstraction with the student's action. For example, if a student exhausted all of the reasons for swimming, the *similar-why-action* dialogue would respond in the following way:

```
Well, that's the only video I have about why animals swim,
but I know about some things that are like swimming. Maybe
one of them will give you a reason you'll like for your
butterfly to swim.
Wading is like swimming, because they are both ways that
animals move in water.
Black herons wade. Black herons wade to lie in wait for prey.
I think this video is funny.
Do you want to see that ?
```

Just as with the *similar-why-feature-dialogue* the search for similar actions is controlled by the property *natural kind* in the *similar-why-action-dialogue.* The algorithm searches up abstraction links from the student's action and then presents the behaviors associated with the actions it reaches and their specializations. In the transcript above, the dialogue manager searched up the *isa* link from *swim* to the action *move in water.* Beneath *move in water* it found *wade* which is associated with the behavior *wait for prey.* Therefore, it suggested *wait for prey* as a reason for swimming. The dialogue manager will continue a search until it reaches the top of the hierarchy or an action with the property *natural kind.* It will not search upward from an action with this property because the actions beyond that point will be too dissimilar from the actions beneath that point to use as suggestions for new behaviors.

The *why-action-dialogue* concludes with a student selecting a behavior that is supported by the student's new action. This behavior may be a new one for the animal or it may be one the animal already pursues. This behavior, in turn, may become the subject of a *how-behavior-dialogue* or a *why-behavior-dialogue.*

6.2.5 How Behavior Dialogue

The *how-behavior-dialogue* tries to find a behavior or action to support a behavior selected by the student. The dialogue manager initiates the *how-behavior-dialogue* when a student chooses a behavior as his initial modification or commits to a new behavior to resolve a *why-action-dialogue* or *why-behavior-dialogue.* Like the *how-action-dialogue,* the *how-behavior-dialogue* may involve a multi-step process in order to satisfy all the conditions necessary for the achievement of a behavior. In Creanimate, the process of determining how a behavior will be achieved parallels the traditional approach to planning in artificial intelligence (Fikes and Nillson 1971; Sacerdoti 1977). The *how-behavior-dialogue* works backward from the desired behavior, assembling the appropriate combination of behaviors and actions necessary to achieve the behavior.

In order to resolve the *How behavior?* question the dialogue manager must traverse the representational tree rooted at the desired behavior. In the process, the student must select individual behaviors that achieve the desired behavior or that are part of that behavior. The student continues to select additional behaviors or actions to support each newly selected behavior until every required aspect of the desired behavior is supported by actions. In other words, the *how-behavior-dialogue* fills in bplans and plans until all of the behaviors involved have been supported by actions. (Once these actions are provided, the dialogue manager will initiate *how-action-dialogues* if necessary to support them.) As an example, we consider the dialogue would ensue from a student's request for a butterfly that can hunt. Figure 34 displays two levels of the the hierarchy rooted at the behavior *hunt.*

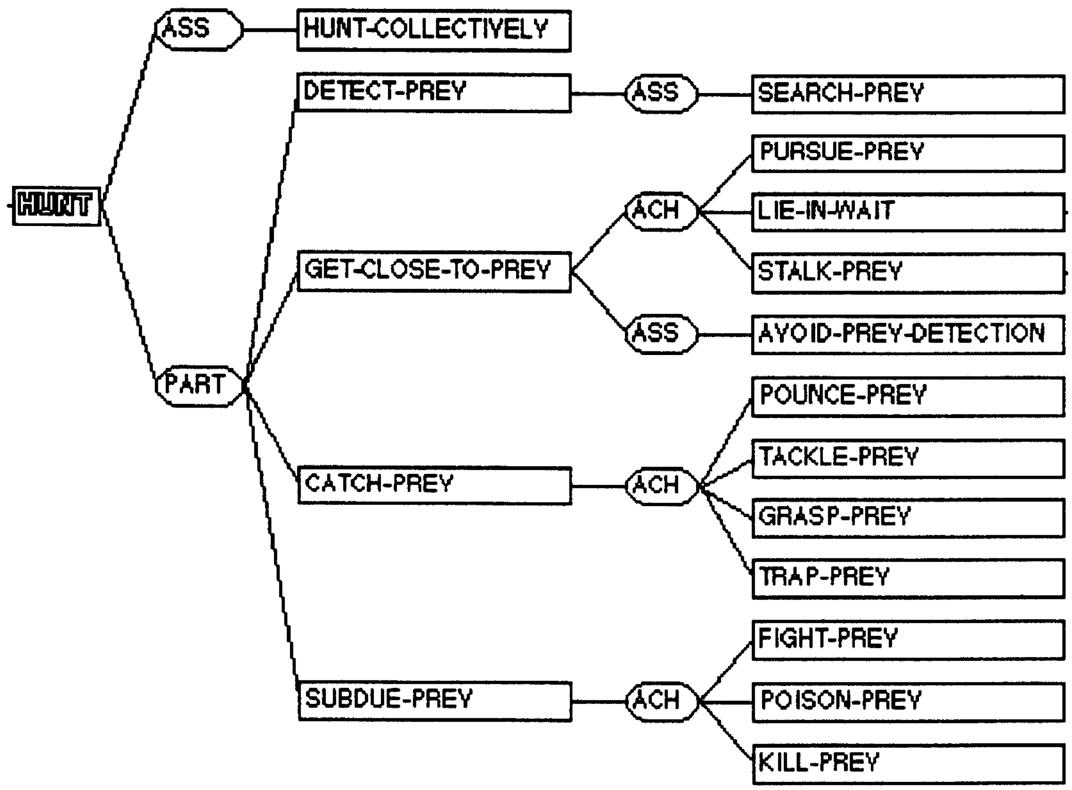

Figure 34. The hierarchy of behaviors that comprise *hunt.* Sub-behaviors are connected with a "PART" link, achieving behaviors with a "ACH" link, and assisting behaviors are connected with a "ASS" link.

A behavior may have either 1) behaviors that achieve it, 2) sub-behaviors, or 3) actions.[3] If a behavior has either behaviors or actions that achieve it, then any one of those behaviors is sufficient to enable an animal to perform that behavior. If it has sub-behaviors, however, then the behavior is a complex one requiring each of the sub-behaviors to achieve it. Every sub-behavior must be supported in order for a behavior to be supported. In addition to actions, achieving behaviors, and sub-behaviors, behaviors may also have assisting behaviors. Assisting behaviors can help an animal to achieve a behavior, but they are neither necessary nor sufficient for its achievement. Therefore, students are free to add assisting behaviors in order to support a behavior, but they do not play a role in satisfying the conditions for the successful achievement of that behavior.

---

[3] These connections are implemented through the slots of behavior frames named *achieved-by, sub-behaviors,* and *actions.* They are described in Chapter 7, The Indexing Vocabulary.

Looking at *hunt,* we see that it has four sub-behaviors, *detect prey, get close to prey, catch prey,* and *subdue prey.* The *how-behavior-dialogue* must resolve each of these behaviors individually. The dialogue manager goes through the *how-behavior-dialogue* in a depth-first fashion, so that it will try to resolve *detect prey* completely before moving on to *get close to prey.* Of course, students are free to use the "Skip this" button in the interface to jump from one sub-behavior to another. When they click the "Skip this" button, they are presented with a choice of other sub-behaviors to work on:

```
One part of hunting is detecting prey.
How would you like your butterfly to detect prey?
Button Press  -SKIP-THIS
We can do any of the things listed below.
Which would you like to do next?
("Talk about how my butterfly will get close to its prey.")
("Talk about how my butterfly will catch its prey.")
("Talk about how my butterfly will subdue prey.")
("Add something else to my butterfly")
("Start a new animal")
("Continue where I was")
```

Assuming the student is content to continue with *detect prey,* the dialogue manager checks its for actions, achieving behaviors, and sub-behaviors. *Detect prey* has no sub-behaviors or achieving behaviors, but it does have a value in its *actions* slot, the action *detect.* This tells the dialogue manager that it is looking for an action to fill in the plan, ... *in order to detect prey.*

```
[behavior detect-prey
     :isa ([behavior find-food] [behavior detect-food] )
     :part-of ([behavior hunt] )
     :assisted-by ([behavior search-prey])
     :actions ([action detect])
     ...]
```

At this point, the dialogue manager poses the explanation question "How would you like your butterfly to detect its prey?" and initiates the standard dialogue cycle looking for an action to support *detect prey.* Any action that appears in the *actions* slot of *detect prey* or is a specialization of a value in the *actions* slot will be accepted as an answer. Thus, a student might choose *see, smell, hear,* etc.

```
[action detect
     :isa ([action use-senses])
     :subtypes ([action hear] [action echolocate]
                [action detect-in-dark] [action smell]
                [action see]
                [action detect-pressure-variations])
     ...]
```

Once the student selects an action to support *detect prey* then the dialogue manager moves on to the next sub-behavior of *hunt, get close to prey*. Unlike *detect prey, get close to prey* has several *achieved-by* values. (These can be seen in the *hunt* hierarchy in figure 34.) Since these values are behaviors, the dialogue manager tries to complete a bplan this time, not a plan. The partial bplan that represents the explanation question is:

```
[bplan :sub ?QUERY :super [behavior get-close-to-prey]]
```

The dialogue manager executes the dialogue cycle to help the student identify a behavior that supports *get close to prey.* We can see from the *hunt* hierarchy in figure 34 that *lie in wait, stalk prey,* and *pursue prey* are all acceptable ways to achieve *get close to prey.* Once the student has committed to one of these behaviors, the dialogue manager will initiate a recursive *how-behavior-dialogue* to identify behaviors or actions that are necessary to support the newly selected behavior.

Through the behavior *hunt,* we have seen each of the three conditions that arise in the *how-behavior-dialogue.* They are summarized below:

1. If the behavior has sub-behaviors, initiate the *how-behavior-dialogue* recursively on each of them.
2. If the behavior has achieving behaviors, conduct the dialogue cycle to allow the student to select one of them. Then initiate the *how-behavior-dialogue* for the newly selected behavior.
3. If the behavior has actions, conduct the dialogue cycle to allow the student to select one of them.

In addition to these three conditions, all of which use the standard dialogue cycle, the *how-behavior-dialogue* also contains an opportunity to consider an animal's current attributes. This occurs at the beginning of the *how-behavior-dialogue* before the dialogue manager starts to traverse the tree of supporting behaviors. When a student adds a new behavior to his animal, the dialogue manager takes the opportunity to discuss the way the animal currently achieves the same goal. Thus, for example, if a student asks for a butterfly that can hunt, the dialogue manager will ask the student how butterflies currently get their food, and the storyteller will show an example reminding if it has one. This question is generated by searching up the hierarchy from the new behavior looking for a behavior that is achieved by the student's animal in the wild. This behavior is called the *intersection behavior.* In the current example, the intersection behavior is *get food* which is achieved both by *hunt* and by *get nectar,* which is a behavior associated with butterflies. The following transcript shows this consideration of how a butterfly gets food. In addition, it demonstrates some of the steps discussed above that arise in the course of creating a butterfly that can hunt.

Suppose you could create a new animal by taking an existing animal and changing it some way. What would you make?

Student: A Butterfly that can hunt.

<u>Consideration of the student's current animal:</u>

Hunting will allow your butterfly to get food. Before we change your butterfly so that it can hunt, we should find out how butterflies currently get food. Do you know how butterflies currently get food?

Student: I don't know

Butterflies drink nectar which is part of getting nectar.
Getting nectar helps butterflies to get food.

Go to first sub-behavior:

One part of hunting is catching their prey.

Pose the explanation question:

How would you like your butterfly to catch its prey?

Student: By biting it

Correct answer example reminding:

That might work. Meerkats bite to grasp prey.
I have a pretty good video about that. This has scorpions in it.
Would you like to see a video about that?

Student: Yes

Similarity-based reminding:

Wow, that meerkat reminded me of a great video.
White breasted sea eagles also grasp prey. Only, instead of biting, they do something different. They grip things.
Is that something you would like to see?

Student: Yes

[VIDEO: "Sea Eagle Catches Fish"]

Identical reminding:

Meerkats are not the only animals that bite in order to grasp prey. For example, bears bite to grasp prey.
This is a great video.
Do you want to see that ?

Student: No

Commitment point:

So your butterfly could bite in order to help it to catch its prey.
You can commit to that now, or we can look at how some other animals catch their prey and you can decide later.
How would you like your butterfly to catch its prey?

Student: Show me more ways

...[Dialogue cycle deleted]...

Another sub-behavior of *hunt:*

Another part of hunting is getting close to their prey.
How would you like your butterfly to get close to its prey?
Student: By flying close
... (Transcript no. Pal-m1-6-11-17.11)

6.2.6 Why Behavior Dialogue

Some behaviors do not have achieving behaviors, sub-behaviors, or actions because they are abstract. Instead they either have subtypes or they are the act of another behavior.[4] For example, the behavior *fight* (displayed in figure 35) is an abstract behavior that has no actions, achieving behaviors, or sub-behaviors. Instead, it is the act of several behaviors including *fight for food, fight predators,* and *fight for mates.* If a student requests an abstract behavior like *fight,* the dialogue manager initiates a *why-behavior-dialogue.* The *why-behavior-dialogue* starts with an abstract behavior and looks for a specialization of it. Once the student has selected such a behavior, the dialogue manager can use this new behavior in a *how-behavior-dialogue.*

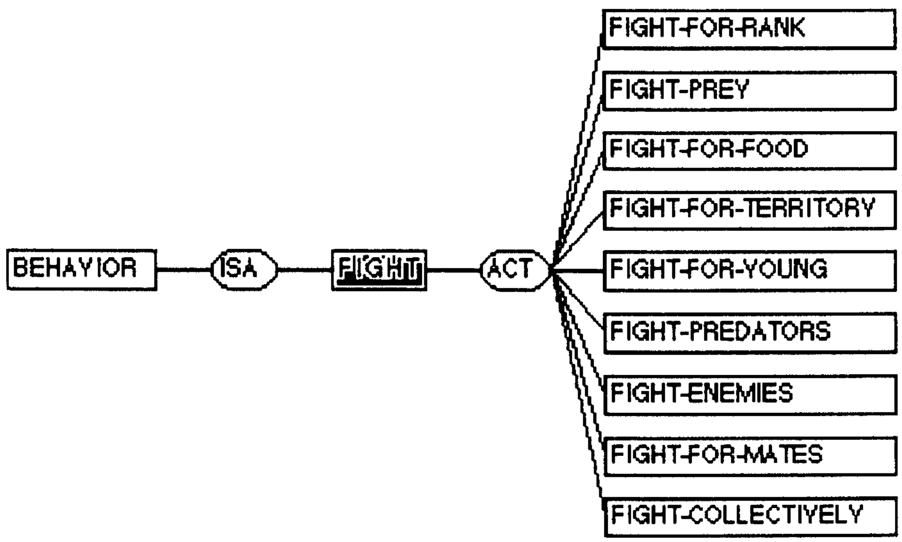

Figure 35. The behavior tree for *fight*

The *why-behavior-dialogue* follows the standard dialogue cycle. Any behavior that is a subtype of the current behavior or has the current behavior as its act is an acceptable behavior. The following transcript in which a student requested a dog that can fight is an example of a *why-behavior-dialogue*:

Pose the explanation question:

`That's a good idea, is there a reason why you want your dog to fight?`

`Student: So it can defend itself`

Correct answer reminding:

`That's a good idea. Bees fight to fight enemies.`
`Would you like to see a video about that?`

---

[4] The *subtypes* and *act-of* links are abstraction links for behavior hierarchies. They are discussed in Chapter 7, The indexing vocabularies.

Student: Yes

[VIDEO: "BEES EXPEL WASP"]

Identical reminding:

```
Bees are not the only animals that fight enemies.  For
example, jawfish fight enemies.
I have a good video about that.  This has battles in it.
Would you like to see a video about that?
```

Student: Yes

[VIDEO: "JAWFISH DEFENDS TURF"]

Commitment point:

```
So your frog can fight to defend itself.
You can commit to that now, or we can look at some other
reasons why animals fight and you can decide later.
Why would you like your frog to fight?
```

... (Transcript no. m4-5-11-11.01)

Once the student settles on a specialization of his behavior, the dialogue manager initiates a *how-behavior-dialogue* to help the student determine how his animal will achieve this behavior.

6.3 Implementing the Dialogue Manager

Managing a dialogue centered around open-ended questions like those in Creanimate is not easy. The dialogue manager must be able to both pose appropriate questions and respond properly to students' answers. The primary duties of the dialogue manager are to:

- Identify and pose relevant explanation questions.
- Evaluate students' answers with respect to the program's knowledge.

In this section, I describe the way in which the dialogue manager performs these duties. The first step in every dialogue is identifying an explanation question that applies to the student's animal and initiating a dialogue about that question. In Section 6.3.1, I describe the algorithm that the dialogue manager uses to identify explanation questions and initiate dialogues about those questions. Once a question has been posed, the next step is conducting a discussion with the student about possible answers to that question. The underlying structure for this discussion is provided by the *dialogue cycle,* which I described above. In order to implement the dialogue cycle in an efficient and general-purpose manner, it was necessary to develop data structures that would support the dialogue manager in asking questions and evaluating answers. These data structures are described in Section 6.3.2.

6.3.1 Initiating Dialogues: Selecting Explanation Questions

The dialogue manager is implemented through a library of dialogue plans, each corresponding to a single explanation question. Each plan[5] is indexed under the explanation question that it applies to. Thus, initiating a dialogue consists of examining the student's animal to identify a relevant explanation question and retrieving a dialogue plan that is

---

[5] Plans are implemented as a body of lisp code. They are not represented declaratively, nor does the dialogue manager have access to their internals as in the traditional AI sense of *plan.*

indexed under that explanation question. The dialogue plan is then instantiated using the particular values from the current explanation question.

The strategy for identifying relevant explanation questions is based on the relationships in the knowledge representation and the objects that participate in those relationships. For every relationship, there are two corresponding explanation questions. For example, feafuns correspond to the questions, *Why feature?* and *How action?* The feafun *fins in order to swim* corresponds to the questions "Why have fins?" and "How to swim?" Similarly, each type of object participates in specified relationships, i.e. features participate in feafuns, actions participate in feafuns and plans, and behaviors participate in plans and bplans. To identify applicable explanation questions, the dialogue manager maintains a record of the modifications that a student has made to his or her animal. Every time the animal is modified in some way, the dialogue manager checks the relationships that the modification can participate in. Based on those relationships, the dialogue manager identifies the appropriate explanation question. This is summarized in table 7.

Table 7.–The identification of explanation questions that apply to a student's animal. Each type of modification (lefthand column) can participate in a limited number of relationships (middle column). Using the student's modification and the relationships that the modification can participate in, the dialogue manager identifies relevant explanation questions (righthand column). Dialogue plans are indexed under these explanation questions.

Modification	Relationship	Explanation Question
Feature	**Feafun**	**Why feature?**
Claws	Claws in order to *action*	Why have claws?
Action	**Feafun**	**How action?**
Dance	*Feature* in order to dance	How to dance?
Action	**Plan**	**Why action?**
Dance	Dance in order to *behavior*	Why dance?
Behavior	**Plan**	**How behavior?**
Fight predators	*Action* in order to fight predators	How to fight predators?
Behavior	**Bplan**	**How behavior?**
Avoid predation	*Behavior* in order to avoid predation	How to avoid predation?
Behavior	**Bplan**	**Why behavior?**
Fight	Fight in order to *behavior*	Why fight?

If a student adds an action to his animal, the dialogue manager recognizes that the action can participate in both feafuns and plans. It also recognizes that the explanation question that applies to a feafun with a specified action and an unknown feature is *How action?* and that the explanation question that applies to a plan with a specified action and an unknown behavior is *Why action?* Therefore, when a student adds a new action to his animal, the dialogue manager creates internal representations of these two explanation questions using the student's animal and the new action. For instance, if a student has been working on a dog, and he changes it so it can dance, the dialogue manager will generate the two explanation questions corresponding to, "How should we change your dog so it can dance?" and "Why will your dog dance?" The first of these is an example of *How action?* and the second is *Why action?*. Indexed under the explanation question category *How action?* are the dialogue plans named *how-action-dialogue* and *check-action-dialogue* and under *Why action?*

is the dialogue called *why-action-dialogue.* (These dialogue plans are described in the next chapter.) When applicable dialogue plans are identified, they are placed on a queue of pending dialogues. As a student's session progresses, the dialogue manager selects dialogue plans one at a time from the the pending dialogue queue.

6.3.2 Asking and Answering Questions

Since questions provide the structure for the dialogue cycle, it follows that questions should also provide the basic organizational unit for the dialogue manager. Questions are represented with a data structure called, simply, a *question.* A question data structure contains information about how to 1) pose a question in natural language, 2) evaluate answers to that question, 3) infer answers to that question from information in the knowledge base and 4) state the answer to the question in natural language. The question data structure is an object in the terminology of object-oriented programming. It packages together four important functions that all deal with a single type of relationship. A question data structure responds to the following messages from the dialogue manager:

Ask: Ask its question in natural language.
Verify: Evaluate a student's answer to that question.
Infer: Infer an answer to the question using the information in the knowledge base.
Say: State an answer to the question in natural language.

Each of these plays an important role in managing open-ended dialogues.

Asking a Question

In response to the **ask** message, a question data structure provides the dialogue manager with a text string that asks the student the appropriate question and a list of appropriate options to allow the student to respond to the question. For example, one of the question data structures in Creanimate is named *why-animal-has-feature.* This data structure provides all of the functionality necessary to handle the question, "Why does *animal* have *feature?*" for any particular combination of animal and feature. If the dialogue manager were to send this question data structure the **ask** message along with the animal *cheetah* and the feature *long strong legs,* the following output would be generated:

```
Do you know what cheetahs use their long, muscular legs for?
```

In addition to generating the question in natural language, the **ask** message to a question data structure also generates a list of options to allow a student to respond to the question. When the question above is asked as a bridge to a story, the options for responding are,

```
So it can ...
Show me a video.
What other videos can I see?
Skip this video.
```

When a student selects an option with an ellipsis such as "So it can ...", he is prompted to type his own answer in place of the ellipsis. Depending on the circumstance in which the question is asked, different options may be presented to the student. If a student saw the question above and chose to see the video, then the question would be repeated following a video. After a video, the options for answering this question would be,

```
So it can ...
Show me the video again.
Tell me.
```

to appear in the absence of a story, the options would simply be:

```
So it can ...
I don't know.
```

When a student selects the options "Tell me" or "I don't know" the dialogue manager generates the answer for him using the **say** message described below.

To summarize, a question data structure responds to the **ask** message with 1) a text string that asks the student the question and 2) a list of options that that allow the student to respond to the question in a manner that is appropriate for the context in which the question is being asked.

Evaluating a Student's Answer

Once the student has provided an answer to a question, the dialogue manager needs to evaluate that answer to determine what to do next. To do so, it sends the **verify** message to the question data structure along with the student's response. The **verify** method evaluates the answer and returns one of the following four values:

Correct: The student's answer matches the information in the knowledge base.

Correct-other: The student's answer is a correct answer for some other animal, but not for the animal in the current question.

Incorrect-but-animal-has: The student's answer is incorrect, but the animal in the question does have the attribute specified in the student's answer.

Incorrect: The student's answer cannot be verified according to the information in the knowledge base.

Correct-other, incorrect-but-animal-has, and **incorrect** are different descriptions of an answer that cannot be verified as correct according to the Creanimate knowledge base. Such unverifiable responses are considered incorrect. For the question *why-animal-has-feature* with the animal *cheetah* and the feature *legs,* the following list shows some possible answers, and the results of evaluating them:

`Do you know what cheetahs use their long, muscular legs for?`

run fast **correct**
Cheetahs do use their legs to run fast.

wade **correct-other**
Cheetahs don't use their legs to wade, but other animals do.

camouflage **incorrect-but-animal-has**
Cheetahs do not use their legs to camouflage themselves, but they do have spots for camouflage.

fly **incorrect**
Cheetahs do not use their legs to fly; there are no other animals that use legs to fly; and cheetahs do not fly in any other way.

The result returned from this evaluation helps the dialogue manager to determine what it should do next. In response to an incorrect answer, the dialogue manager will request an incorrect answer reminding from the storyteller. Depending on whether the evaluation is **incorrect, incorrect-but-animal-has,** or **correct-other**, the text generated by the dialogue manager will vary appropriately. These variations are described in the discussion of the **say** method below. In response to a correct answer the dialogue manager will confirm the student's response and request a correct answer reminding from the storyteller.

To summarize, the **verify** method of a question data structure is used by the dialogue manager to evaluate the correctness of a student's answer.

Inferring the Answer to a Question

A question data structure contains the functionality to allow the dialogue to infer answers to questions itself, by searching the knowledge base. Suppose the dialogue manager needed to know what action cheetahs use their legs for. The **infer** routine for the question *why-animal-has-feature* would search up the hierarchy starting from the specified animal looking for a feafun containing the given feature. For the cheetah, it would find the feafun, *strong long legs in order to run fast* and it would return the action from that feafun, namely *run fast*. Because of speed considerations, the **infer** method is not used in the most recent version of Creanimate.

Stating the Answer to a Question

In addition to information for asking questions and evaluating answers, the question data structure also contains information on how to make statements. In response to the **say** message, a question data structure will generate an appropriate statement using a specified animal and other concepts. For example, the **say** message to the question *why-animal-has-feature* with the animal *cheetah,* the feature *strong-long-legs* and the action *run fast* generates the following statement:

```
Cheetahs use their long, muscular legs to help them to run
fast.
```

The **say** message method is used in many different circumstances in dialogues. First, it is used to construct bridges to example stories.

```
Bees dance. Bees use their tails to help them dance.
I think this video is amazing.
Is that something you would like to see?
```
(Transcript no. pal-m1-6-30-12.40)

The sentence, "Bees use their tails to help them dance," was generated by the **say** message to the question *how-animal-does-action* using the animal *bee,* the action *dance,* and the feature *tail.* However, as we have already seen, many stories are introduced with questions not statements. In those situations, the student responds to the question either by seeing the video or providing an answer. If he chooses to see the video, he sees the question again immediately following and has the option to review the video, answer the question, or respond, "tell me." In each of these circumstances, the **say** message gets called. The following examples show the range of answers that a student might give, and the resulting use of the **say** method. The first four examples correspond to the four possible evaluations returned by the **verify** method discussed above. The last example shows a student asking the system for the answer.

Student answers the question correctly (**correct**):

```
Sage grouses have wings. They have feathered wings. Do you
know what sage grouses use their wings for?
(I have a hilarious video about that. )
Student:   So they can dance.
Yes, that is right. Sage grouses have wings to help them to
dance.
Would you like to see a video about that?
```

Student answers the question incorrectly (**incorrect**):

```
...Do you know what sage grouses use their wings for?...
Student: so they can dig
I don't know any animals that have wings to help them to dig.
But I do know sage grouses have wings to help them to dance.
If you want, I can show you some animals that dig.
```

Student answers the question incorrectly (**correct-other**):

```
...Do you know what sage grouses use their wings for?...
Student: so they can fan themselves
Actually, sage grouses have wings to help them to dance.
But other animals have wings to help them to fan themselves.
For example, Fox bats have wings to help them to fan
themselves. We can see that in a moment.
```

Student answers the question incorrectly (**incorrect-but-animal-has**):

```
...Do you know what sage grouses use their wings for?...
Student: so they can run
Sage grouses do run. But, sage grouses have wings to help
them to dance.
Is that something you would like to see?
```

Student asks for the answer:

```
...Do you know what sage grouses use their wings for?...
Student: show me a video
[VIDEO: GROUSE DANCES]
Now, do you know what sage grouses use their wings for?
Student: tell me
Sage grouses have wings to help them to dance.
```

In the same way that the list of answers returned by the **ask** method can be tailored to the context, the output generated in response to a **say** message can be adapted to context. The **say** method for a question data structure accepts data structures called *impacts* as optional arguments in order to tailor the output. (Impacts are described in Chapter 7.) In the following transcript, the relationship *strong hind legs in order to kick dirt* was passed to the **say** method along with the impact *surprising*. This leads the dialogue manager to indicate that the relationship is surprising when it generates the statement.

```
Kangaroo rats use their strong hind legs for a surprising
reason:  to kick dirt.
```

Say methods can take other optional arguments to generate output such as the following, in which the optional argument **also** was given:

```
Thompson's gazelles also use their long legs to help them run
fast.
```

Thus, **say** methods are used by the dialogue manager to generate natural language statements from a question data structure. These functions generate specific sentences using the animal and attributes passed to them by the dialogue manager. Optional arguments to the **say** methods can produce special output for certain circumstances.

The Importance of the Question Data Structure

As we've seen, the question data structure serves as a repository for information about discussing a particular relationship in the course of a dialogue. The question data structure tells the dialogue manager how to ask a question about a relationship, evaluate the answer, infer answers on its own, and state that an animal possesses that relationship. The use of question data structures was important for the construction of the Creanimate system because it allowed us to write general-purpose routines to execute steps in the dialogue cycle regardless of which dialogue plan was being executed or which particular explanation question was being discussed. Using this approach, special-purpose information about specific questions could be encoded in the question data structures, while the routines that manipulate these data structures could be general-purpose. As a result, most of the dialogue cycle could be implemented through general-purpose functions that were shared by all of the individual dialogue plans.

6.4 Summary

The Socratic dialogue manager conducts dialogues structured around important explanation questions for the domain of animal adaptation. The dialogues are designed to take advantage of a student's investment in the animal that he creates as a motivator for him to examine issues for that animal's adaptability. In the traditional Socratic style, the dialogue manager responds to a student's design with thought-provoking questions that help him to explore these issues through his own hypotheses.

The dialogue manager is implemented as a library of dialogue plans that are indexed according to the explanation questions they discuss. Dialogue plans are initiated when the explanation question they discuss become relevant to the student's animal. These dialogue plans all share a basic structure, provided by the dialogue cycle. The dialogue cycle consists of four steps: 1) asking the explanation question, 2) evaluating a student's answer, 3) presenting example remindings, and 4) allowing the student to commit to an answer. In addition, individual dialogues may elaborate on this structure with a step in which the animal's original attributes are considered and with an extension to the dialogue that considers answers to similar explanation questions.

The six dialogue plans in the dialogue manager's repertoire start with a modification that a student has made to his animal. The dialogues all follow the dialogue cycle in which the dialogue manager poses an explanation question and the student considers possible answers. The goal of each dialogue is to have the student relate the modification he has made to his animal to some other feature, action, or behavior in a way that would help the animal to survive in the wild. Each dialogue has specific criteria for acceptable answers and provides opportunities for the students to learn from relevant stories.

The general dialogue algorithm is supported by data structures that encode knowledge about handling specific questions in the course of dialogues. These question data structures store information about how to ask students questions, how to evaluate their answers, how to infer answers, and how to state answers declaratively.

Chapter 7

The Indexing Vocabulary

The storyteller in a case-based teaching system requires an indexing vocabulary to label the stories in its library. Every story must have an index that describes the situations in which the story is relevant. An index is composed of elements from the indexing vocabulary. In the course of a dialogue, the storyteller examines these indices in its search for stories that are relevant to a student's situation. To support this story retrieval, the indexing vocabulary must be expressive enough to describe the range of storytelling situations that arise in the task environment. While the primary role of the indexing vocabulary in a case-based teaching system is the support of indexing and retrieval, the indexing vocabulary in Creanimate also plays a critical role in supporting the dialogue manager. The representation language that provides the indexing vocabulary also enables the dialogue manager to draw the inferences about animal adaptations necessary to manage its open-ended question-and-answer dialogues. The indexing vocabulary in Creanimate supports four major activities of the storyteller and dialogue manager:

- Story retrieval (storyteller)
- Dialogue initiation (dialogue manager)
- Evaluation of student responses to questions (dialogue manager)
- Communication (dialogue manager and storyteller)

To support these activities, the Creanimate indexing vocabulary is capable of expressing both the specific information about stories required by the reminding strategies of the storyteller and the general information about animal adaptation employed by the dialogue manager in conducting dialogues. These capabilities are provided by the Creanimate knowledge representation language. This knowledge representation language is a slot-filler database implementation of a semantic network (Quillian 1966: Charniak et al. 1987). It is designed not only to support the indexing requirements of the storyteller and the inferencing requirements of the dialogue manager, but to provide the ability to generate and interpret natural language.

In this chapter, I describe the components that make up this representation language. In particular, I focus on the representational challenges that arose in the construction of this knowledge representation language and the solutions we developed. In the next chapter, I will describe the way the indexing vocabulary is used to construct indices and the indexing tool that we have developed to enable a human indexer to manage the complexity of a large knowledge base.

7.1 Design Considerations for the Indexing Vocabulary

Two important principles guided the design of the indexing vocabulary. Both principles follow from the primary objective of this research, which is to build effective educational systems. Both principles place decisions concerning representation at the service of this primary objective. These principles are:

- Minimal representation.
- Primacy of pedagogy

7.1.1 Minimal Representation

Representation is the critical issue in this research. Creanimate succeeds or fails with the success or failure of its representation scheme. If the representation were inadequate or the vocabulary too limited, Creanimate would not be able communicate effectively with students or retrieve stories when they are relevant. In Creanimate, unlike other research in AI that focuses on the content issues of representation, e.g., CYC (Guha and Lenat 1990; Lenat and Guha 1990), the representation is not the end goal. In this case, the end goal is educational. Since the representation exists solely in service of the educational goals, our objective has been to represent the domain as little as possible in order to achieve those goals. We have adopted this principle of minimal representation because the primary research is concerned with the quality of the system's performance as an instructional system, not the quality of its internal architecture on a scale of elegance. Where issues concerning the representation have appeared in the course of conducting this research, our question has never been, "What is the correct way to represent these concepts?" but, "What is the simplest representation we can have for these concepts and still achieve the goals of the system?" This approach is essential because "correct" representation is a Holy Grail. No matter how close a representation gets to being correct, there is always another intricacy to represent. If we had tried to represent the issues for animal adaptation "correctly" we would have had to solve the entire AI problem.

One of the advantages of case-based teaching over other AI approaches to teaching systems is that a case-based teaching system need not understand everything it is teaching. Unlike the traditional ITS approach to instruction (Sleeman and Brown 1982; Wenger 1987), the teaching system does not need to understand the domain as well as the student does. Traditional ITS's must understand the domain well enough to maintain a model of a student's current understanding of the subject matter and to improve that understanding. A case-based teaching system only needs to understand its domain well enough to monitor a student's actions in the task environment and produce stories to help the student learn from his situation. Most of the knowledge that the student gains from a case-based teaching system is either provided by the structure of the task environment or is encapsulated within stories. This architecture enables the system to teach topics that are elaborate and complex, indeed well beyond the current state of the art of representation in AI, without requiring a representation capable of handling that complexity. Since this complex information can be provided by stories, the teaching system is left with the sole responsibility of getting the right story to the student at the right time. Therefore, the burden of understanding for a case-based teaching system is to understand the actions that a student can take in the task environment in a way that will enable it to find the right story for that student. This does not require that the system be capable of understanding or representing much of the information that is contained within stories.

Minimal representation is an essential approach if one is to avoid the explosion of intricacies that arise in any knowledge representation effort. The value of the case of the case-based teaching architecture is that with only a minimal representation a case-based teacher is able to convey knowledge that is much richer and more complex than can be captured by the representation itself.

7.1.2 Primacy of Pedagogy

In any AI application, the choices made in the design of the representation language reflect a theory of the domain. In a teaching system this domain theory plays an important role in shaping a student's experience with the system. Since the decisions made in developing the representation language influence what a student will learn, it is imperative that the pedagogical goals direct the development of this representation. This imperative constitutes the principle we call *primacy of pedagogy*. In Creanimate, the decisions made in designing the indexing vocabulary show up in students' interactions in three forms: 1) the particular explanation questions that the dialogue manager poses; 2) the ways in which it interprets and evaluates students' responses to these questions; and 3) the specific contexts which the storyteller recognizes as appropriate for a particular story.

As we saw in Chapter 2, explanation questions convey the important relationships of a domain. Therefore, in Creanimate, determining the appropriate explanation questions was the first step towards developing a representation and indexing vocabulary. This choice of explanation questions was based on an analysis of what is appropriate and important for students in the target age group to understand about animal adaptation. The choice of these explanation questions in turn determined which types of concepts and relationships had to be included in the knowledge representation. For example, historically the first explanation question implemented in Creanimate was *How action?* e.g., "How would you like to change your tortoise so it can run fast?" This explanation question reflects an educational goal that students recognize that every action an animal performs requires a set of enabling physical features. Furthermore, students should know the particular features required for particular actions. To present knowledge to the student in this form, Creanimate must have its knowledge organized in the same fashion.

Therefore, each explanation question establishes important requirements for the Creanimate knowledge representation. For instance, the implementation of *How Action?* requires that the knowledge representation include a vocabulary to describe physical features, actions, and the different ways that actions can depend on physical features. In addition, it must provide a way of associating a particular feature with a particular action for a particular animal. In Creanimate these requirements gave rise to the objects called *features, actions,* and *feafuns*. The implementation of this explanation question also influenced the structure of indices. For the storyteller to retrieve stories that are relevant to a discussion of the *How Action?* explanation question, the indexing vocabulary must be able to indicate that a particular story illustrates the use of one or a set of physical features to support an action.

Maintaining the primacy of pedagogy means using pedagogical goals to shape a teaching system's representation. In the development of Creanimate, the pedagogical goals were specified first, in the form of explanation questions. The representation language was then designed around these questions.

7.2 The Creanimate Knowledge Representation: An Overview

The Creanimate knowledge representation language is a frame language[1] composed of two frame types: objects and relationships. Objects are used to represent single concepts and

---

[1] For readers who are unfamiliar with this terminology, there is a good introduction to frame representation languages in Charniak et al. (1987). For a good historical perspective on the development of semantic networks, frame languages and other issues in knowledge representation see Brachman and Levesque (1985).

relationships are used to link two or more objects to a third object. As introduced in Chapter 5, the types of objects and relationships are:

Object Types	**Relationship Types**
Animal	Feafun
Feature	Plan
Action	Bplan
Behavior	Rule
Phys-obj	
Property	
Impact	
Index	

Each type of object and relationship has a corresponding class of frame defined with a fixed set of named slots. The fillers for these slots are other objects and relationships. For example, frames that represent animals have slots for the features, actions, and behaviors that those animals possess. Likewise, features, actions, and behaviors all have a slot that contains a list of animals that they are associated with. Thus, conceptually the Creanimate knowledge base is an extensive network of nodes (frames) connected by named links (slots). Many of the links in the knowledge base are bi-directional, meaning that if an animal has a link connecting it to a feature then there is a complementary link from the feature back to the animal. Bi-directional links enable the inferencing mechanisms used by the storyteller and dialogue manager to conduct flexible and efficient searches.

Abstraction

For efficiency in both representation and inference, objects in the Creanimate knowledge base are organized into abstraction hierarchies. The animal hierarchy, for example, has the most general category (called simply *animal*) at the top, and specific species and subspecies of animals at the bottom. In between are increasingly specific categories of animals, some of which coincide with the categories in the phylogenetic tree and some of which are *ad hoc* categories that suit the pedagogical goals of the system. The categories in abstraction hierarchies are not mutually exclusive, so one object may fall into several overlapping categories. Therefore, an object may have several parents. That means that abstraction hierarchies are directed acyclic graphs, not simple trees. Displayed below is a portion of the hierarchy that shows the parents and children of the animal *primate:*

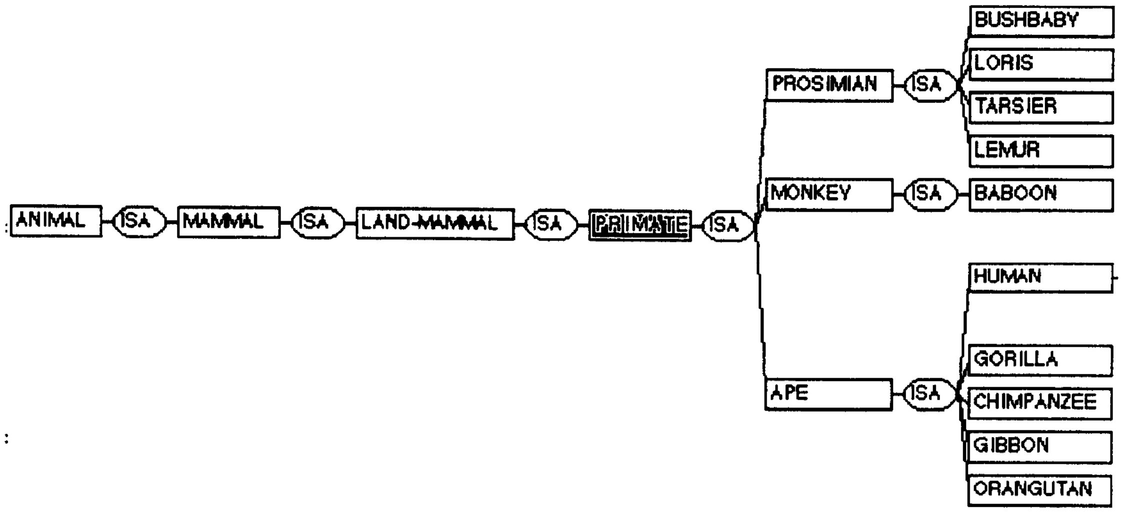

Figure 36. The hierarchy showing the parents and children of *primate*. The other children of *land-mammal, mammal,* and *animal* are not shown.

While animals have only one type of abstraction link, the *isa* link, other object classes have several types of abstraction links, each of which carries different semantics. Abstraction hierarchies serve two main purposes. They allow the dialogue manager to identify similarities among concepts that share parents, and they implement inheritance.

Inheritance

Inheritance of attributes through abstraction hierarchies provides Creanimate with an important efficiency in representation. When an object is connected to a parent by an abstraction link, then the child inherits certain attributes of the parent. Thus, for example, *primate* inherits the feature *warm-blooded* from *mammal*, and *bushbaby* inherits the feature *tail* from *prosimian*. The underlying frame system allows for flexible inheritance, so that different slots may be inherited across different abstraction links. In addition, slots may be inherited from multiple parents across more than one type of abstraction link. To implement this multi-parent, selective inheritance, each slot for each object class has a set of abstraction links that are designated as that slot's *inheritance links*. The inheritance links for a slot may be all the abstraction links for the object class, or they may be a subset of them. (The inheritance links for a particular slot in an object are an attribute of the object's class, e.g., animal, not the object itself.) Inheritance provides the primary mechanism for drawing inferences in Creanimate.

The combination of bi-directional links and inheritance in Creanimate, requires that some values be inherited from children instead of parents. This *inverse inheritance* is necessary for the following reason. Bi-directional links are made up of two individual links called *forward links* and *inverse links*. If a forward link is inherited from parents to children then symmetry requires that the corresponding inverse link must be inherited from children to parents.

Inheritance provides significant savings in both the amount of memory required for the knowledge representation and in the amount of effort necessary to construct the knowledge representation. Inheritance allows an indexer to link one object with another at a high level of abstraction and provides a mechanism for the system to infer the same link between two objects that inherit from them.

7.3 Representation Issues in Creanimate

In accordance with the goal of minimal representation, the representation language in Creanimate started out with the fewest types of objects possible. Initially there were only four object types: animals, features, actions, and behaviors. Each object had only one abstraction link, the *isa/subtypes* link, to connect it to other objects of the same type. Animals had links to connect them to each of the features, actions, and behaviors that they employ. Features were connected with bi-directional links to the actions they support, just as actions were connected to the behaviors they support. So, initially, the frames used to represent objects had only the following slots:

Animal	**Feature**	**Action**	**Behavior**
Isa/Subtypes	Isa/Subtypes	Isa/Subtypes	Isa/Subtypes
Features	Animals	Animals	Animals
Actions	Actions	Features	Actions
Behaviors		Behaviors	

This very simple representation scheme provided many of the capabilities required by Creanimate, but it fell short of fully supporting the needs of either the storyteller or the dialogue manager in a number of respects. For example, some necessary inferences could not be drawn because the semantics of the slots were too vague. Some other inferences were not possible because they required access to relationships that were not be captured by the available slots. In response to these shortcomings, the representation was extended through a series of incremental improvements. Each improvement was designed to address a specific problem that hindered the program's ability to achieve its pedagogical goals. These problems fell into five categories of representational challenges:

1. Representing the requirements of performing actions and behaviors.
2. Representing specific forms of similarity.
3. Representing relationships involving more than two different types of objects.
4. Recording subjective impressions.
5. Generating and interpreting natural language.

In the following sections, I discuss the specific examples of these problems that arose in the development of the Creanimate knowledge representation. For each problem, I also describe how the knowledge representation was enhanced to resolve it. As a supplement to this discussion, the reader may want to refer to Appendix A, which contains a summary description of the slots in all of the objects and relationships in the Creanimate knowledge representation.

7.3.1 Expressing the Requirements of Objects

The initial, minimal Creanimate representation supported the asking of why questions, i.e. "Why would an animal have a beak?" but it did not really support the asking of how questions. The initial links enabled the dialogue manager to look at a feature or an action and determine what it was used for, but it did not allow it to look at an action or a behavior and determine what combinations of actions or features were necessary for an animal to perform the action or behavior. To implement dialogue plans that discuss how questions, it was necessary to add slots to describe requirements of actions and behaviors.

Requirements of Actions

For an animal to perform an action, that animal must have features that support that action. To breathe requires lungs or gills, and to run requires legs. However, some actions are complex. They are performed through a combination of more primitive actions. In the initial, minimal representation, the *features* slot made it possible to indicate that a feature was used in some way to perform an action, but it was not possible to determine which particular combinations of features were required to perform the action. In addition, it was impossible to represent an action as the composition of several other actions.

> **Issue:** Different features may be combined in several ways to perform an action.
> **Example:** Swimming fast requires 1) one of a number of features to propel the animal, e.g. fins or a tail, and 2) a sleek body shape.
> **Solution:** *Requires/required-by* and *suffices/suffices-for* slots.

The *requires/required-by* and *suffices/suffices-for* slots in actions and features. In addition to the initial *features* slot in actions and *actions* slot in features, features and actions have two additional slots that indicate more precisely the ways that particular features support particular actions. Any feature that appears in the *features* slot of an action must also appear in either the *requires* or *suffices* slot of the action.

The *requires* slot of an action contains a list of features which are all necessary to perform the action. In order for a student's animal to perform an action, it must have each of the features in that action's *requires* slot or specializations of those features. The *suffices* slot of an action contains a list of features, any one of which is sufficient to enable an animal to perform that action. In other words, the *requires* slot indicates that a combination of features are necessary, and the *suffices* slot indicates that only one feature is necessary. Any particular action may only have values in one or the other of the *requires* or *suffices* slots. Since features can be combined into groups (see below), there is some flexibility in the use of these slots. A group in the *suffices* slot means the same thing as each of the features appearing individually in the *requires* slot. Therefore, if there are several ways to perform an action, each requiring a conjunction of features, each conjunction should appear in the *suffices* slot as a group.

The action below, *hide in shell,* is an example of the use of the *requires* slot.

```
[action hide-in-shell
     :isa ([action hold-self-in-position  ]
           [action take-refuge  ] )
     :animals (  [animal turtle  ] )
     :behaviors ( [behavior defend-against-predators  ] )
     :features ( [feature shell  ]
                 [feature retractable-body-parts  ] )
     :requires ( [feature shell  ]
                 [feature retractable-body-parts  ] )
    ...]
```

Since *shell* and *retractable body parts* both appear in the *requires* slot of *hide in shell,* the dialogue manager knows that an animal can only perform the action *hide in shell* if it has both a *shell* and *retractable body parts.* In the frame for the action *smell,* shown below, the features *nose, nostrils,* and *antennae* all appear in the *suffice* sslot which tells the dialogue manager that any one of those features is sufficient to enable an animal to smell.

```
[action smell
     :isa ([action use-senses][action detect]
           [action investigate])
     :animals ([animal bear][animal pike]
               [animal tree-cricket])
     :behaviors ([behavior detect-insects  ]
                 [behavior detect-vegetation  ]
                 [behavior locate-carcass  ]
                 ...)
     :suffices ([feature nose  ]
                [feature nostrils  ]
                [feature antennae  ] )
     ...]
```

The *requires* and *suffices* slots are used by the *how-action-dialogue* plan to determine if the physical features that a student has given to an animal are sufficient to enable the animal to perform a desired action.

Issue: A feature necessary to perform an action may be composed of several individual features.
Example: To hide in a shell like a turtle requires retractable body parts. This group of features is composed of retractable legs, a retractable head, and a retractable tail.
Solution: *Group/part-of* slots

The *group/part-of* slot in features. The *group* link is used to bring together several different features that all work together to perform an action as if they were a single feature.

```
[feature retractable-body-parts
     :group ([feature retractable-legs  ]
             [feature retractable-head  ]
             [feature retractable-tail  ] )
     :animals (  [animal turtle  ] )
     :actions ( [action hide-in-shell  ] )
     :required-by ( [action hide-in-shell  ] )
     ...]
```

Because *retractable body parts* is really a group composed of *retractable legs, head,* and *tail,* it inherits any attributes that each of them may have individually. However, since they are all required together in order to *hide in its shell,* they are associated together in the group. The *group* slot is an abstraction slot, so if an animal has the feature *retractable body parts,* the system is able to infer that it also has each of the elements of the group individually. The inverse link for *group* is *part-of.*

In making sure that a student's animal has all of the features to perform an action, the *check-action-dialogue* checks to see if any of the features has parts. If one of the features does, the dialogue manager will initiate a dialogue in which the student has the opportunity to add any necessary parts of the feature to his animal.

Issue: One action may be performed through the simultaneous or sequential performance of several more basic actions.
Example: Flying is really composed of several actions, taking off, generating lift, steering in flight, and landing.
Solution: *Parts/part-of* slots.

The *part-of/parts* slot in actions. When an action is performed through a combination of several other actions, these actions appear in the *parts* slot of the action. For example, *take off, steer, generate lift,* and *land* are all in the *parts* slot of the action *fly*. While a parent across a *part-of* link is considered an abstraction of its child, there is no inheritance across *part-of/parts* links.

```
(DEFFRAME action fly
     :isa (       [action move-long-distance  ]
                  [action move-fast  ]
                  [action move-in-air  ]
                  [action move-above-ground  ] )
     :subtypes ( [action soar  ] )
     :parts ([action take-off]
             [action generate-lift]
             [action steer-in-flight]
             [action land-from-flight])
     ...)
```

In the *check-action-dialogue* the dialogue manager checks each of the parts of an action to determine whether a student's animal has all of the features necessary to perform the action.

Requirements of Behaviors

In the initial, minimal Creanimate representation, behaviors were linked to other behaviors through *isa* hierarchies and to actions through the *actions* slot. The *actions* slot works well for indicating the requirements of some behaviors, such as *pursue prey* or *attract mate.* These behaviors can be achieved through one of a number of actions, e.g., *run fast* or *dance.* However, high-level behaviors, e.g., *eat,* cannot be linked directly to any actions because they are more complex. Instead, they are achieved by other behaviors, either individually or in combination. Therefore, the representation of behaviors was enhanced to include a *goal achievement* hierarchy. The goal achievement hierarchy connects more specific behaviors to the high-level behaviors that they achieve. This hierarchy follows *achieves, part-of,* and *assists* links. The goal achievement hierarchy supplements the *isa* hierarchy, which captures other aspects of similarity.

Issue: A high-level behavior may be achieved by one of several other behaviors, instead of being directly achieved by any specific action.
Example: Getting food can be achieved by hunting, foraging, scavenging, etc.
Solution: *Achieved-by/achieves* links.

The *achieved-by/achieves* slots in behaviors. The *achieves* link indicates that performing the child behavior is a way to successfully achieve the goal of the parent. In general, the difference between one behavior and the behavior that it achieves, is that either the *act* of the behavior or its *object* is more specific. For example, *maintain warmth* has a more specific object than *maintain body temperature.* Since, by definition, a behavior can only serve one survival goal, it can only have one behavior in its *achieves* slot. However, a behavior may have an unlimited number of behaviors in its *achieved-by* slot. The *achieves* link is an inheritance link, so attributes of behaviors are inherited across *achieves*.

The *achieves/achieved-by* links are used by the *how-behavior-dialogue* plan to identify behaviors that can be used to achieve a student's selected behavior.

Issue: One behavior may be achieved through several, sequential behaviors.
Example: The behavior *hunt* involves four other behaviors: *detect prey, approach prey, catch prey,* and *subdue prey.*
Solution: *Sub-behaviors/part-of* slots.

The *sub-behaviors/part-of* slots in behaviors. The *sub-behaviors/part-of* slots are used for the representation of behaviors that consist of several individual behaviors. A behavior may only be part of one other behavior. In addition, a behavior may have a value in either its *part-of* or its *achieves* slot, but not both. Figure 37 shows the sub-behaviors that comprise the behavior *hunt*.

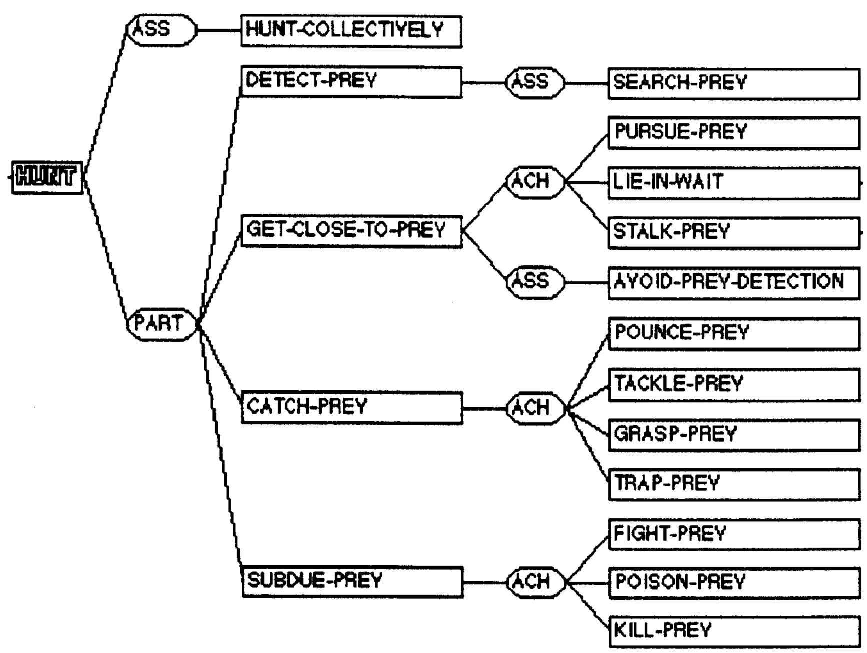

Figure 37. The behavior hierarchy beneath *hunt*. *Part-of/sub-behavior* links are labelled "part"; *assists/assisted-by* links are labelled "ass"; *achieves/achieved-by* links are labelled "ach".

During the *how-behavior-dialogue* the dialogue manager checks each of a behavior's sub-behaviors to insure that a student's animal is able to achieve a behavior he has selected. The student has the opportunity to add the actions necessary for his animal to perform each of the sub-behaviors.

Issue: A behavior may play a role in achieving another behavior but may be neither necessary nor sufficient for the achievement of that behavior.
Example: Avoiding detection can be helpful in getting close to prey, but it is neither necessary nor sufficient.
Solution: *Assisted-by/assists* links.

The *Assisted-by/Assists* slots in behaviors. A behavior that can help an animal to achieve another behavior but is not essential for achieving that behavior is connected to it via an *assists* link. The reasons *assists* is used in the case of *detect prey* and *get close to prey* is that it is possible to get close to one's prey making any special effort to conceal oneself. However, many animals do engage in the behavior *avoid prey detection* to help them perform the behavior *get close to prey.* Since a behavior that assists with one behavior will also assist an animal to achieve its children, *assisted-by* links are inherited.

7.3.2 Expressing Specific Aspects Of Similarity

While the *isa* relationship encodes non-specific similarity between objects that share a parent in many cases, it is necessary to represent more specific forms of similarity. For example, behaviors may be similar because they share the same underlying activity or because they share the same goal. Depending on which form of similarity two behavior possess, they will be treated differently in dialogues about achieving behaviors. Similarly, for features, it is useful to indicate whether two features share some structural similarity, e.g. size or shape, or whether they are two versions of the same anatomical structure, e.g. a nose. As a result, the vague semantics of *isa* links were divided into more specific categories of similarity, expressed through more specific links.

Issue: The *isa* relationship among features indicates that they both correspond to the same anatomical structure. However, it is also important to represent the fact that two features share some structural similarity.
Example: Tongues, trunks, and some tails all share the property that they are flexible.
Solution: *Qualities/quality-of* slots.

The *qualities/quality-of* slots of features. The *qualities* link connects an aspect of a feature, called the quality, to a feature itself. For example, *red color* is a quality of the feature *red breast.* Since *red color* has the functionality *be visible, red breast* inherits this functionality from *red color.* Qualities include color, size, and shape. The following is a list of representative qualities:

```
[feature strong]                [feature long-quality  ]
[feature flat]                  [feature hard  ]
[feature short-quality]         [feature soft  ]
[feature big-quality  ]         [feature small-quality  ]
[feature curved]                [feature flexible  ]
[feature noticeable]            [feature unnoticeable  ]
[feature fat]                   [feature skinny  ]
[feature blue-quality]          [feature black-quality  ] )
```

Issue: While *isa* effectively represents the fact that one action achieves the same end as another, it is useful to represent that two actions are similar in some other respect.
Examples: Swim fast and run fast share the quality of moving fast.

Solution: *Qualities/quality-of* slots.

The *qualities/quality-of* slots of actions. Just as with features, the *qualities* slot links an attribute of an action with specific actions that share that attribute. A quality generally expresses an adverbial modifier. Thus, *fast* might appear in the *qualities* slot of both *run fast* and *swim fast.* Qualities can also be used to represent the locations that actions take place, such as *under water.*

Issue: While *isa* represents the fact that two behaviors share a specialization of both their act and object, many behaviors only share the same act.

Example: *Detect prey* and *detect food* both share the same act, *detect,* and the same object, *food.* However, *detect predator* and *detect prey* only share the same act.

Solution: *Act/act-of* slots.

The *act/act-of* slots in behaviors. The *act* slot of a behavior contains a behavior that is a more abstract description of the activity involved in the first behavior. This means that behaviors that share the same parent by an *act* link share a basic similarity in the way that they are performed. For example, the behavior *flee* is the *act-of* both *flee predator* and *flee cold.*

```
(DEFFRAME behavior flee
     :isa ([behavior move])
     :act-of ([behavior flee-predator]
              [behavior flee-enemies]))
```

By definition, a behavior must be goal-directed. Therefore, the value of the *act* slot must have a goal, however its goal may be less strictly specified than the goal in the child behavior. In fact, acts are a special case of behaviors, in that they may be general enough to encompass more than one survival goal, e.g., *flee.* As a result, the only place in the behavior hierarchy that such a behavior may exist is in the *act* slot of another behavior. Because a behavior that appears in the *act* slot of another behavior is permitted to have a vague goal, it is not allowed to have any values in its *achieves* slot. In other words, if a behavior is the *act-of* another behavior (either directly or by inheritance), then it may not achieve or be part of some other behavior. A behavior is only allowed to have one value in its *act* slot. Values of the *act* slot are inherited across *achieves* and *isa* slots.

The *act/act-of* links are important because students frequently request behaviors with vague goals, e.g. *fight.* When a student requests a behavior like *fight* that is an *act-of* other behaviors, the dialogue manager initiates the *why-behavior-dialogue* plan in order to identify a goal for that behavior. For example, a student might choose *fight predators* or *fight other males* as a reason for fighting.

An important form of similarity for features is similarity of location in an animal's anatomical structure. This similarity, represented through attachment, was added to accommodate the vague manner in which people use language. For example, in answer to the question, "Do you know what bears use to swat other animals?" students might respond, "their paws," "their claws," or "their arms." While any one of these answers indicates that the student understands the answer, the knowledge base typically has an action like swatting associated with only one feature, the most specific one involved. Therefore, it is important that the dialogue manager understand the relationship between each of these features in a way that enables it to accept one feature for the other in students' answers. The link that is used in Creanimate to indicate this relationship is called *attachment*. Attachment differs from composition (represented by *group/part-of* links) because attachment represents similarity of location in the anatomical structure, whereas composition represents a mutual interdependence for the performance of an action. The same features can, however, be related through both attachment and composition.

> **Issue:** It is necessary to represent the connection between features that are attached to each other. In conversation, these terms can be used loosely to refer to each other.
>
> **Example:** A paw is attached to a leg.
>
> **Solution:** *Attached-to/attachment* slots.

Attached-to/Attachment. A physical connection between features is represented by the *attached-to/attachment* links. A feature that is farther away from an animal's body is *attached-to* a feature that is closer. So, *leg* would appear in the *attached-to* slot of *foot*.

```
(DEFFRAME feature feet
     :isa (       [feature end-of-appendage  ] )
     :subtypes (  [feature feet-with-toes  ]
                  [feature feet-without-toes  ] )
     :attached-to ( [feature legs  ] )
     ...)
```

The *attached-to* relationship is useful for interpreting student answers. In response to the question, "Do you know what a cheetah uses to bite its prey?" a student might respond, "its mouth." Although, the dialogue manager might be seeking *sharp teeth,* it can infer that the student's answer is compatible with the desired answer using the *attached-to* relationship . In a situation like this, the dialogue manager would respond, "That is right. Cheetahs use their sharp teeth to bite."

7.3.3 Relationships Involving More Than Two Objects

In the initial Creanimate representation, objects representing animals had links to each of the features, actions, and behaviors that the animals possess or engage in. However, since features can support several actions, actions can support several behaviors, and vice versa, these links were not sufficient. For example, bears have both teeth and claws. They also perform the action *scrape*. With only links to individual features, actions, and behaviors, the dialogue manager could not infer whether the bear used its teeth, its claws, or some other feature in order to scrape. Furthermore, if an index indicated that a story contained *teeth, claws,* and *scrape,* the storyteller could not determine if the story was an example of teeth scraping or claws scraping. It is clear that both the dialogue manager and storyteller require a representation that can express in specific terms, which features support which actions and which actions support which behaviors, as well as which behaviors are supported by which other behaviors for a particular animal or index. This problem is solved using frames called *relationships*. Individual relationships are not named, nor are they connected by abstraction hierarchies.

Feafuns, Plans, and Bplans

Issue: The representations for animals and indices must indicate which of the features that an animal possesses are used to perform which of its actions.
Example: A bear could use either its teeth or its claws to scrape.
Solution: The relationships called *feafuns*.

A feafun is a relationship that links together a feature with an action it supports. Feafuns appear in the *feafuns* slots of animals and indices, as in the example below:

```
(DEFFRAME animal red-cockadoo-woodpecker
     :isa (      [animal flying-bird  ] )
     :features ( [feature small-beak  ] )
     :actions (  [action peck  ] )
     :feafuns (  [feafun :feature [feature small-beak]
                         :action [action peck]] )
     ...)
```

Issue: The representations for animals and indices must indicate which of the actions that an animal engages in are used to support which of its behaviors.
Example: A beaver could either swim or run to flee predators.
Solution: The relationships called *plans*.

Plans link an animal with a specific action that it uses to achieve one of its behaviors. For example,

```
[plan :action [action swim]
      :behavior [behavior flee-predator]]
```

Issue: A particular animal can use one of several behaviors to achieve another behavior.
Example: An animal could get food by hunting, foraging, scavenging, etc.
Solution: The relationships called *bplans.*

A plan has two slots called *sub* and *super.* The behavior in the *sub* slot is used by the animal with the bplan in order to help the animal perform the behavior in the *super* slot. The bplan *hunt in order to get food* is displayed below:

```
[bplan :sub [behavior hunt] :super [behavior get-food]]
```

The behavior in the *sub* slot of a bplan must be connected to the behavior in the *super* slot by either an *achieves, assists,* or *part-of* link. Because a behavior can only support one other behavior through *achieves* or *part-of* links, bplans are not strictly necessary for behaviors connected in these ways. If an animal performs a behavior that achieves, assists or is part-of another behavior, then the bplan that connects the two can be assumed for that animal. Therefore, in most cases, bplans are treated as implicit by Creanimate rather than being explicitly represented. Bplans are important, however, for consistency in the general-purpose reminding and dialogue management routines that deal with feafuns, plans, and bplans identically.

The feafun, plan, and bplan relationships are best viewed as multi-directional links. Instead of connecting one object to another the way traditional slots do, these relationships connect one object with two others. Feafuns connect an animal to both a feature and an action, plans connect animals to both actions and behaviors and bplans connect animals to two behaviors. They allow the dialogue manager and storyteller to recognize not just which features, actions, and behaviors an animal might have or a story might illustrate, but the pairs of these objects that go together for the particular animal or story.

Rules

In addition to feafuns, plans, and bplans, Creanimate contains a fourth form of relationship that is required by the expectation-violation reminding strategy. The expectation-violation reminding strategy requires the ability to represent standard expectations. These expectations are generalizations that a student might expect to be true but are not, in fact, true in all cases. Standard expectations are represented using the relationships called *rules*. Rules differ from the rest of the knowledge representation in the fact that they express information that is not universally true. Each rule in Creanimate is violated in an interesting way in some story. Rules are used to trigger remindings about these exceptions.

> **Issue:** Representing standard expectations.
> **Example:** Only birds have beaks. (In fact, many mollusks, including octopuses, have beaks.)
> **Solution:** The relationships called *rules*.

There are three types of rules in Creanimate, all-rules, only-rules, and no-rules. Each type of rule is used to represent a different type of generalization. Like the other relationships, rules do not have any abstraction hierarchies or inheritance.

The following example is a rule that tells the storyteller that the generalization, "All birds use their wings to fly," is violated by a story showing octopi.

```
[only-rule :animals ([a bird])
           :value [feature beak]
           :index [index octopus-spar]
```

This particular rule triggers an expectation-violation reminding any time a student asks to give a beak to an animal other than a bird. The slots in rules connect animals to attributes that those animals are either expected to possess (all-rules and only-rules) or not to possess (no-rules) and to the indices of stories that violate those expectations.

7.3.4 Recording Subjective Impressions

The success of Creanimate depends in large part on how natural the dialogue seems to the student. An important part of a natural-seeming dialogue is output that is capable of conveying subjective impressions. To enable the system to produce this output, an indexer records his own subjective impressions of stories and the relationships that they contain. These opinions appear in the bridges and other statements described in Chapters 5 and 6. They are recorded in the form of objects called *impacts*. Their name comes from the fact that they are intended to convey the impact that a story or statement will have on a student.

> **Issue:** Presenting subjective impressions of the relationships in stories that are depicted in stories makes dialogues more natural and helps to interest students in seeing those stories.

Example: The way kangaroo rats use their hind legs to kick dirt in the eyes of predators is surprising.

Solution: Associating *impacts* with the relationships that appear in stories.

The *impact* slot in relationships is used by the storyteller in generating the bridges that introduce a story. The following feafun with the impact *surprising* appears in the index for a story about kangaroo rats. In this story, a kangaroo rat kicks dirt in the eyes of predatory snake:

```
[feafun :feature [feature strong-hind-legs]
        :action [action kick-dirt]
        :impact [impact surprising
                        :object [action kick-dirt]]]
```

The impact causes the storyteller to introduce this story in the following fashion:

```
Kangaroo rats use their strong hind legs for a surprising
reason: to kick dirt.
I have an amazing video about that.
Would you like to see that?
```

In Chapters 5 and 6, we saw questions used as bridges to stories. The decision that a particular relationship in a particular story should be introduced by a question is a subjective decision made by an indexer. In general, the criteria for an askable relationship in an index are: 1) the animal must be familiar to nearly all the students in the target population, and 2) some students should know the answer.

When the impact *askable* is associated with a relationship, that tells the storyteller that a story depicting that relationship can be introduced with a question. Below is the feafun *talons in order to grip* that contains the impact *askable* from a story about eagles.

```
[feafun :feature [feature talons]
        :action [action grip  ]
        :impact ([impact askable])]
```

The presence of the *askable* impact in this feafun causes the storyteller to generate the following bridge when introducing the story it appears in:

```
White breasted sea eagles have claws.  Their claws are called
talons.  Do you know what white breasted sea eagles use their
claws for?
(I have a great video about that. )
Would you like to see that?
```

An impact only appears in a relationship when the relationship is part of an index.

Issue: Giving a student someone's overall impression of a story can help promote the story.

Example: A particular student might especially like stories with chase scenes in them and dislike stories with close-ups of insects.
Solution: Associating story impacts with indices.

Indices contain a slot for storing impacts that describe the story as a whole. This *impacts* slot contains an impression of the story that an indexer feels will help the student to make an informed judgement about whether or not he wants to see the story. In many cases, this impact serves as an advertisement highlighting elements that make the story most appealing. Just as with impacts stored in relationships, impacts associated with indices themselves are used to construct the bridges that introduce stories. The impacts that apply to indices fall in the five categories described below. Each index may have several impacts but only one of each type.

Impression. An impression impact describes the way a viewer will react to a story. Some impressions are *funny, surprising, gruesome,* and *exciting.*
Sample output: `I think this video is exciting.`

Action. An action impact describes the action that appears in a story. Some action impacts are, *high-speed chase, graceful swimming.*
Sample output: `I like this video because it has graceful swimming in it.`

Animal. An animal impact describes the animals other than the animal in the index that appear in the video. For example, a story about a school of fish that also includes a shark might have the animal impact *shark.*
Sample output: `If you like sharks, you'll love this video.`

Technique. A technique impact describes the way a video was filmed. Some technique impacts are, *close-up* and *slow-motion.*
Sample output: `This video was filmed in slow-motion.`

Location. A location impact describes the place that a video was filmed. Some location impacts include *underwater, in the treetops, in a beehive.*
Sample output: `They had to be underwater to film this video.`

Impacts are used in both relationships and indices to enable the storyteller to introduce stories in the way that a person would. The introductions to stories produced by Creanimate contain not just objective information about the contents of the story and how it relates to the student's discussion but also subjective information about what is exceptional and appealing about a particular story.

7.3.5 Controlling Search

In each of the reminding algorithms the storyteller searches through the knowledge base examining the indices that are associated with animals or their attributes. For the example reminding and expectation-violation reminding algorithms, controlling these searches is not difficult because they only search up or down a hierarchy. Because these algorithms only search in one direction, they terminate either when they find a story or when they reach the top or bottom of the hierarchy. However, the similarity-based reminding algorithm is implemented as a widening search that goes both up and down abstraction hierarchies. Left unchecked, this spreading activation will search an entire abstraction hierarchy. Therefore, it was necessary to introduce a mechanism to prevent this similarity search from spreading beyond objects that are reasonably similar to the starting point. As I discussed in Chapter 5 the Creanimate system has used two different techniques for controlling this search.

Issue: Establishing boundaries for similarity searches.
Solution: The *abstraction* slot.

***Abstraction* slots in feafuns and plans.** Like the *impact* slot, the *abstraction* slot is only used in relationships that appear in indices. The *abstraction* slot in a relationship contains another relationship made up of objects that, individually, are abstractions of each of the objects in the initial relationship. For example, a story about a skunk squirting skunk juice contains the feafun *scent duct in order to squirt noxious fluid.* In the abstraction slot of that feafun appears the feafun *orifice in order to propel fluid.*

```
[feafun :feature [feature scent-duct]
        :action [action squirt-noxious-fluid]
        :abstraction [feafun :feature [feature orifice]
                             :action [action propel-fluid]]]
```

This abstraction conveys the information that any story that shows an orifice being used to propel fluid in some way could be considered similar to the skunk story. The abstraction is used by the similarity-based reminding algorithm to identify similar stories. The relationship in the *abstraction* slot establishes the boundaries of a set of stories that can be considered similar to the current story. For more details see the discussion of the original similarity-based reminding algorithm in Chapter 5.

Issue: Controlling similarity search with abstractions requires an indexer to add information to every feafun or plan in every index specifically for this purpose rather than using general-purpose information from the knowledge representation.
Solution: *Functionality/functionality-of* slots.

The *Functionality/functionality-of* slots in features/actions. The *functionality* slot connects features to actions. This slot is only used in abstract features. An abstract feature is one that appears as an internal node in the feature hierarchy and would never be directly linked to an animal. The *functionality* relationship expresses a similarity of function among the children of an abstract feature, as opposed to simply a similarity of structure. For example, in Creanimate the feature *sharp* is a quality of *sharp teeth, quills,* and *talons,* among others. *Sharp* is an abstract feature which would never be directly associated with any animal. It provides, however, a useful category for gathering together several features which share sharpness as a basic quality. The *functionality* slot represents the usefulness of a feature's being *sharp*. *Sharp* has the abstract action *pierce* in its *functionality* slot. The *functionality* slot is used by the improved similarity-based reminding algorithm described in Chapter 5.

The *abstraction* and *functionality/functionality-of* slots both allow the storyteller to establish bounds on its similarity search. Abstractions require less processing at run-time, however they require additional work by the indexer as he enters stories into the system. The *functionality* slot, coupled with the other slots that link features to actions, allow the storyteller to dynamically generate the same information available in the *abstraction* slot by using general-purpose information from the knowledge base.

7.3.6 Generating and Understanding Natural Language

In conducting dialogues with students, the Creanimate system must be able to both generate natural-sounding prose and interpret students' free-form input. Creanimate generates output by combining sentence templates with text strings that correspond to objects in the knowledge base. These text strings are produced from slots called *say* slots that are attached to objects.

> **Issue:** Generating natural language output corresponding to objects in the knowledge base.
> **Example:** Producing the question, "Do you know why sage grouses dance?" using the objects `[animal sage-grouse]` and `[action dance]`.
> **Solution:** *Say* slots.

Say slots are used to generate the natural language equivalent to a frame. The system possesses basic rules for conjugating verbs and converting the number of nouns. However, in order to deal with exceptions, specific slots that correspond to the different ways that a concept may appear in an English sentence are necessary.

Generating Noun Forms from Animals and Features

Since features and animals are nouns, the system must be able to generate natural language equivalents in both singular and plural. Because features may be expressed as phrases it is necessary to allow for exceptions when a feature appears following a possessive noun. These conditions are handled by the following slots:

Say1. The *say1* slot is used to talk about a single animal. For an animal, the *say1* value is the singular form of the animal's name. Thus, the *say1* value for *cheetah* would be "cheetah." For a feature, whether the *say1* value is the singular or plural form depends on whether an animal has one or several of the feature. For *nose,* the *say1* value would be "nose" because an animal only has one nose. However, for *feet,* it would be "feet," since an animal has more than one foot. The Creanimate natural language generator contains rules for producing the values for the other *say* slots from the value of the *say1* slot. However, in the case of irregular nouns, an indexer overrides the rules by adding explicit values to the appropriate slots.

Say. The *say* slot contains the text string that corresponds to multiple animals. The *say* slot always contains a plural form. For example, the *say* value for *cheetah* is "cheetahs" and the *say* value for *nose* is "noses." Since most sentences that Creanimate generates use the plural, the *say* slot is the most commonly used form. In the absence of a value in the *say* slot, Creanimate will generate it from the value in the *say1* slot.

Say-its and **Say-their.** The *say-its* and *say-their* slots only apply to features. They are used for features that have a *say* value that is a phrase. For example, the animal *manta ray* has a feature called *manta wings.* These fins have the *say* value "wings that are like fins." This phrase works well in the sentence, "Manta rays have wings that are like fins." However, it does not sound natural in the sentence, "They use their wings that are like fins to swim." Therefore, the *say-their* slot is used for *manta wings* with the value, "wing-like fins." When the *say-their* value is used, the sentence above appears as, "They use their wing-like fins to swim."

Generating Verb Forms from Actions and Behaviors

Since behaviors and actions represent verbs, the system must be able to conjugate them for number and tense. In addition to the simple present form, the system uses the present participle, and the infinitive. As with nouns, the natural language generator contains rules for generating other forms from one form, the plural, third person, present form. This value appears in the *say* slot. Therefore, the other slots are only present in order to record exceptions. The *say* slots that are used in actions and behaviors are:

Say and **Say1.** Just as with nouns, the default number is plural. Therefore, the *say* value is the default. The *say* slot contains the third person plural present form of the verb. For example, the *say* value for *run* is "run." The *say1* value can usually be generated by the system from the *say* value according to standard rules. The most common instance when generating the *say1* value from the *say* value will not work is with behaviors that are generated as phrases. For example, *defend itself* has the *say* value of "defend themselves" and the *say1* value "defend itself."

Say-ing, Say-ing1, Say-to, and **Say-to1.** The *say-ing* and *say-ing1* slots are used to store the present participle form for an action or behavior. The *say-to* and the *say-to1* slots contain strings corresponding to the infinitive. These slots are only used for verbs that are exceptions to the standard rules.

Interpreting Student Input

An important strength of the Creanimate system is that it allows students to enter their own ideas in their own words. However, understanding natural language places a heavy burden on the system. To respond appropriately to natural language input, the dialogue manager must be able to interpret student input in terms of the concepts in its knowledge base. In other words, it must be able to translate students' phrases typed in free text into elements of the indexing vocabulary. After three decades of research, understanding natural language remains an unsolved problem in artificial intelligence and was not a problem that we wanted to solve as just one part of developing the Creanimate system. Therefore, to meet our pedagogical objectives, the system had to be designed to allow students the freedom to express themselves in free text without requiring us to solve the general natural language problem.

What makes natural language understanding manageable in Creanimate is the design of the user interface. Rather than allowing students to enter full sentences in response to the program's questions, we provide them with partial sentences that they complete. Without this interface, we would have had to devote significantly more resources to natural language understanding without realizing any gains in the performance capabilities of the program. For example, consider the range of possible sentences that a student might construct as an answer to the question, "How would you like to change your butterfly?" He might say any of the following, all of which should produced the same response from Creanimate.

```
I want it to have a big nose.
Give it a big nose.
A big nose.
Can I give it a big nose?
Let's give it a big nose.
```

Rather than devote energy to the problem of understanding all the different ways to express "Give an animal a feature" and distinguish it from all the ways to express, "Give it an action" and "Give it a behavior," we decided to provide students with partial sentences to complete. Therefore, in response to the question, "How would you like to change your butterfly?" a student sees the following options:

```
Give it ...
So it can...
So it is ...
```

These partial sentences ease the natural language understanding problem for Creanimate in two ways. They remove the need to handle all of the different ways to construct a sentence with the meaning, "Give it a big nose," and they make it easier to distinguish sentences that mean "Give it a feature," from sentences that mean "Give it an action," or "Give it a behavior." When a student selects the options, "Give it..." or "So it is ..." Creanimate can assume that the phrase the student types in will be a feature. Similarly, if he selects "So it can..." then the program can expect either an action or a behavior. The ability to rely on these expectations greatly reduces the difficulty of the natural language processing task in Creanimate. For example, Creanimate does not need to handle the ambiguity of the word "fly" between the animal *fly* and the action *fly* because it always knows whether it is looking for an animal or an action to fill in a particular blank.

Thus, the natural language understanding task in Creanimate is to map from a phrase entered by a student to an object in the knowledge base. The class of that object is predetermined by the partial sentence that the student selects. In other words, natural language understanding in Creanimate is a matter of recognition. Whenever a student enters a phrase, the dialogue manager attempts to recognize that phrase as one of the concepts in the knowledge base. Martin (1989; 1990) describes an architecture for natural language understanding called the Direct Memory Access Parser (DMAP) which is ideally suited to Creanimate. A DMAP parser is a recognition-based parser that operates on hierarchically organized memories like Creanimate's knowledge base. In a DMAP parser, text strings are associated with individual concepts in the knowledge base, and the parser uses the links between concepts to traverse the knowledge base as it processes a phrase. As it processes a partially parsed phrase, it builds up predictions about the remaining elements of the phrase. These predictions are generated by examining the links that connect concepts in memory. Predictions make the DMAP parser efficient and flexible in its ability to handle missing or unrecognizable information in an input.

In Creanimate, the DMAP parser works in the following way. Suppose, Creanimate asks a student why he wants his turtle to run fast, and the student selects "So it can..." and then enters:

```
chase after female turtles
```

From the question which is of the form *Why action?* and the choice of "So it can..." the parser gets the initial prediction that the input is a behavior. It processes the phrase one word at a time from left to right. The constituent "chase" leads to the activation of two behaviors, *pursue* and *drive away* (i.e. "chase away"). The parser then traverses the specialization links associated with these behaviors in order to generate predictions to guide further processing. The behavior *pursue* has two behaviors in its *act-of* slot, *pursue-prey* and *chase mates. Drive away* has the behavior *drive away enemies* in its *act-of* slot. The parser therefore predicts that the remainder of the phrase will match one of these more specific behaviors. If it doesn't then the parser will return one of the two general behaviors. As it continues to process the phrase, the parser's predictions for *chase mate* match against "chase after female turtles," and the parser reports to the dialogue manager that it has recognized *chase mates* in the student's input.

As this brief example shows, the DMAP architecture is ideal for Creanimate because the parsing task is one of recognition and because the structure of Creanimate's knowledge base is hierarchical. To support the DMAP parser, two forms of information had to be added to the knowledge base. First, text strings were associated with the appropriate concepts in the knowledge base. Second, information was added to tell the parser which links to traverse through the knowledge base in the course of processing input. The Creanimate parser is described in greater detail in Fitzgerald (1992).

Issue: Providing the DMAP parser with the information necessary to generate predictions and recognize phrases.

Solution: *Dmap-parses* slot.

Creanimate objects have a single slot devoted to storing the information required by the parser. The *dmap-parses* slot stores one or several "phrases" that the parser uses to process input and generate predictions. These phrases combine text strings, actual objects from the knowledge base, and the names of slots. These elements can combined in arbitrary permutations. An object may have any number of phrases in its *dmap-parses* slot. The following examples show the range of possible parser phrases that could be associated with the behavior *pursue prey:*

`pursue prey`: Activated by the presence of "pursue" in a student input. Predicts the string "prey" as an upcoming constituent of the input.

`[behavior pursue] prey`: Activated when a constituent of the student's input leads to the activation of the behavior *pursue.* Predicts the string "prey" in the remainder of the input.

`[behavior pursue] [animal prey]`: Activated by the activation of the behavior *pursue.* Predicts the presence of any of the constituents that would activate the animal *prey.*

`:act :object`: Activated by the activation of any of the behaviors in the *act* slot of *pursue prey* (The behavior *pursue* is the *act* of *pursue prey.*) Predicts the presence of constituents that will activate any concepts in the *object* slot of *pursue prey.* (The animal *prey* appears in the *object* slot of *pursue prey.*)

7.3.7 Minimal Representation Revisited

In response to specific needs of the Creanimate system, the knowledge base has been enhanced. However, the guiding principle has continued to be minimal representation. The knowledge representation was expanded only in response to specific needs and only with minimal modifications. Specifically, the knowledge representation was improved in the following ways:

- New links between features, actions, and behaviors were added to indicate the manners in which they support each other.
- A new type of frame called a relationship was created in order to link more than two objects together.
- Information was added to the hierarchies to help control the search that is central to the reminding strategies.
- Objects were created to store the intuitive impressions that allow the storyteller to construct natural bridges when it introduces stories.
- Information was added to assist in the generation and understanding of natural language.

A summary of the resulting knowledge representation can be found in Appendix A. This appendix shows the slot names and a description of their fillers for all of the classes of objects and relationships in the Creanimate knowledge representation.

7.4 Unresolved Representation Issues

In the introduction to this thesis I emphasized the fact that an important goal of this research was to provide a testbed for investigating reminding strategies and exploring pedagogical issues. Therefore, it has been an important priority of this research to reach a point at which reminding strategies could be implemented and pedagogical effectiveness evaluated as quickly as possible. In achieving those objectives it has been necessary to limit our goals concerning the range of relationships taught. It was important that the representation be broad enough to be educationally valuable but that the representation task not be overly ambitious. Therefore, explanation questions about important areas of animal adaptation have not yet been implemented. As the program is extended, however, these will be integrated into the existing system. Some of the outstanding representation issues are described below.

Habitat. The habitat that an animal lives in plays a crucial role in determining which survival goals are in jeopardy for that animal and what strategies are available to achieve those goals. For example, an animal that lives in the tropics may have to face the issue of maintaining a low enough body temperature, but it will not have to deal with maintaining warmth. Habitat will probably be represented as a collection of individual features including available food sources, climactic conditions, and the animal populations that share the habitat.

Interaction Among Animals. In the current system, it is not possible to represent the impact on both animals of an action or behavior that involves two animals. For example, the behaviors *attract a mate* and *select appropriate mate* are both represented in Creanimate. Yet it is not possible to represent the interaction between a male's *attract mate* behavior and a female's *select appropriate mate*. The same is true of *pursue prey* and *flee predator*. Combined with habitat factors, the ability to represent interactions among animals will enable Creanimate to make inferences about whether a selected strategy for getting food will be effective for the food sources or prey populations in a particular animal's environment. To make inferences like this, however, there must be a way to represent the interaction between the activities of both individual animals and of animal populations.

Interference among animal attributes. The current representation of features, actions, and behaviors represents affordance relationships, but not interference. For example, Creanimate currently knows that there are several ways to enable a flying animal to stay aloft, all of which involve having wings that are large enough for the animal's body size. However, this is represented as positive information about what combinations of features enable the animal to stay aloft. It is not possible to represent the fact that being heavy directly interferes with flying. Likewise, the system currently cannot know that having lungs is incompatible with having gills. To make inferences about how changes to animals will interfere with an animal's current abilities, the system must be able to reason about the way that attributes interfere with each other.

7.5 Summary

In any case-based teaching system, the indexing vocabulary supports the indexing and retrieval of stories for the storyteller. In Creanimate, however, the indexing vocabulary also supports the dialogue manager. It provides the general knowledge about animals and their attributes necessary to manage open-ended dialogues about students' animals. The indexing vocabulary, called the Creanimate knowledge representation, was designed in accordance with the principles of minimal representation and primacy of pedagogy. These principles ensure that the instructional objectives of the teaching system take priority in every design decision.

The Creanimate knowledge base is a semantic network that links animals together with their attributes. This network is implemented using frames with fixed sets of slots. Specific frame types are used to represent animals, physical features, actions, and behaviors. These frames are organized into abstraction hierarchies with inheritance of attributes. Beyond associating animals with their attributes, the knowledge representation language was enhanced to respond to five sets of representational issues: representing the requirements of actions and behaviors, expressing various forms of similarity, recording subjective impressions, controlling search, generating and interpreting natural language.

Chapter 8

Indices and the Indexing Tool

In this chapter, we turn from the indexing vocabulary itself to a discussion of how it is used to index stories. In the first section of the chapter, I describe the structure of indices and how their structure supports Creanimate's reminding strategies. In the second section, I discuss the role of the indexer. An indexer is a system developer who does not program, but is responsible for maintaining the knowledge base and creating the indices for stories. In the third section, I describe the Creanimate Indexing Tool. The indexing tool is a software environment designed to support a human indexer with the process of building and maintaining the knowledge base, as well as indexing video clips. The indexing tool has been a vital resource for managing the complexity of the knowledge base as it grew to include over a thousand different features, actions, and behaviors.

8.1 The Structure and Use of Indices

The Creanimate knowledge representation supplies the vocabulary for describing the contents of a story in an index. An index in Creanimate is an object implemented in the same fashion as the other objects. Indices are frames with slots, most of which contain objects from the Creanimate knowledge representation. Each index describes the role of one animal in the story. Any particular story may have several indices. Thus, a story about a robin chasing a cat away from a nest full of chicks might have three indices, one for the robin, one for the cat, and one for the chicks. The following example shows an index for a story about a woodpecker using its beak to probe for insects. This index is relatively simple; some more elaborate indices can be found in Appendix C.

```
(DEFFRAME index woodpecker-pecks-woodpecker
     :animals ([animal red-cockadoo-woodpecker])
     :features ([feature small-beak])
     :behaviors ([behavior search-for-insects])
     :actions ([action peck])
     :feafuns ([feafun :feature [feature small-beak  ]
                       :action [action peck  ]
                       :abstraction
                             [féafun :feature [feature beak]
                                     :action [action pierce]]
               ])
     :plans ([plan :action [action peck  ]
                   :behavior [behavior search-for-insects  ]
```

```
                    :impact ([impact askable])
                    :abstraction
                        [plan :action [action pierce]
                              :behavior [behavior get-food]]])
        :story [story woodpecker-eats]
        :impact ([impact cool  ]
                 [impact slow-motion])
        :rating 8
        :VIDEO ((101001 102500 (disk 'nat-and-brit) "A")))
```

8.1.1 The Primary Slots in Indices

Indices have two parts. The first part describes an animal and the attributes that it displays in the story, and the second describes some other aspects of the story. The slots devoted to describing the animal in the story are:

Animal. An animal that appears in the story.
Features. Features used by the animal in the story.
Actions. Actions performed by the animal in the story.
Behaviors. Behaviors achieved by the animal in the story.
Feafuns. Feafuns demonstrated by the animal in the story.
Plans. Plans demonstrated by the animal in the story.

Except for the *animal* slot, these are the same slots that appear in an animal frame. In fact, the values that appear in these slots in an index also appear in the same slots in the frame for the animal. However, the values in the slots of an index are the subset of the animal's total attributes that actually appear in the story.

The *animal, features, actions,* and *behaviors* slots are bi-directional slots that link indices directly to objects in the knowledge representation. The storyteller traverses these links in the course of searching for stories about particular objects. So, for example, in a discussion of how to fight, the storyteller would start at the behavior *fight* and then search down abstraction links to other behaviors. At each behavior, it would examine the indices linked to that behavior to see if it is appropriate the current context. The bi-directional links that connect objects to indices enable the storyteller to identify candidate indices in this way for both example remindings and similarity remindings. These candidate indices must then be compared with the target concept for the current reminding strategy.

Candidate indices are evaluated by comparing the relationships that appear in the index with the target concept for the current reminding algorithm. As I discussed in Chapter 5, the target concept in a reminding algorithm is always a feafun, plan or bplan. For example, if a student proposes that his animal fight by stabbing other animals, the target concept would be:

```
[plan :action [action stab] :behavior [behavior fight]]
```

Searching down from the behavior *fight* the storyteller would find the behavior *fight enemies*.

```
(DEFFRAME behavior fight-enemies
     :isa (       [behavior defend-against-enemies  ] )
     :achieves ( [behavior protect-self  ] )
     :act (       [behavior fight  ] )
     :animals (  [animal jawfish  ] [animal bee  ] )
     :stories (  [index bees-stinging-enemy-bee  ]
                 [index bees-fight-wasp  ]
                 [index jawfish-defends-territory  ] )
     ...)
```

This behavior is linked to three indices that all appear in its *storie* slot. (The *stories* slot in a behavior is the inverse link of the *behaviors* slot in an index.) All three of these indices become candidate remindings. Since the student proposed that his animal stab other animals in order to fight, the storyteller must examine each of these indices to see if they contain this target concept. One of these indices, the one named *bees-stinging-enemy-bee,* contains the following three plans in its *plans* slot:

```
[plan :action [action sting] :behavior [behavior fight-enemies]]
[plan :action [action drag]  :behavior [behavior fight-enemies]]
[plan :action [action swarm] :behavior [behavior fight-enemies]]
```

Since *stab* is an abstraction of *sting,* through *isa* links, and *fight* is an abstraction of *fight enemies* through *act* links, the target concept matches against the plan *sting in order to fight enemies* from the story.

As this example shows, the slots in indices that describe the attributes of an animal in a story support the example and similarity-based reminding algorithms in two ways. First, the direct links to the objects in the knowledge base enable the storyteller to locate candidate indices by traversing hierarchies of animals, features, actions, and behaviors and identifying objects that are linked to indices. Second, the slots for feafuns, plans, and bplans enable the storyteller to match the contents of indices against the target concept for the current reminding strategy.

8.1.2 The Secondary Slots in Indices

The remaining slots in an index describe other aspects of a story. They are:

Expect-viols. *Expect-viols* is short for "expectation violations." The *expect-viols* slot contains rules that are violated by a story.

Impacts. Subjective descriptions of the story.

Rating. A numeric value between 1 and 10 that the storyteller uses to decide between two stories that are relevant. The rating is intended to capture a story's overall appeal.

Story. The *story* link is a pointer to a story object.

Video. The *video* slot is used to store the physical address of the video clip on a disk.

The *expect-viols* slot supports the expectation-violation reminding strategy by providing a direct bi-directional link to rules representing expectations that are violated by the story. When the storyteller identifies a rule that applies to the current discussion, it traverses this link to the appropriate index. In the *impacts* slot, the storyteller finds the impacts it uses to construct bridges for stories. The *rating* slot gives the storyteller information about the relative value of stories. A higher rating indicates that a story is more dramatic, exciting, or unusual. The storyteller gives preference to stories with higher ratings when all other factors are equal. The *story* slot links an index to a story object. The role of a story object is to connect together all of the indices that describe the same video clip. This prevents the storyteller from showing the same story twice as a result of retrieving it through two different indices. The final slot, the *video* slot, is used by the storyteller to locate the actual video clip on a videodisk. The *video* value specifies the disk on which the clip appears,as well as the frame numbers of the first and last frame in the clip.

8.1.3 The Indices in Creanimate

The current Creanimate knowledge base includes 206 indices. However, the prototype version tested in the schools had access to only 60 minutes of video, which limited it to 135 indices. The indices in Creanimate cover a wide range of birds, mammals, fish, reptiles, and amphibians. A partial list of those indices follows.

```
jawfish-defends-territory
bees-make-honey
ostrich-dances-ostrich
bear-gets-salmon-disk
salmon-jump-salmon
geese-migrate-canadian-geese
nurse-shark-and-remora-nurse-shark
lion-sets-territory-lion
mom-and-baby-chimp-chimpanzee
leopard-carries-food-into-tree-leopard
anteater-sniffs-anteater
ox-digs-in-snow-musk-ox
cricket-smells-tree-cricket
assassin-bug
baby-vulture-asks-for-food
skunk-and-wolf
eagle-catches-rabbit
squirrels-flee-predator
rabbit-escapes-eagle
elephants-at-waterhole-elephant
wildebeests-migrate-wildebeest
gerenuks-eat-and-fan-selves-gerenuk
robin-and-cat
grouse-dances-sage-grouse
```

Creanimate is able to get a great deal of mileage out of the clips that it possesses because each clip can be used in several different ways. In the story selection process, versatility is emphasized. The versatility of a clip is determined by the number of animal attributes that can be recorded in the index for that clip. For example, an index with five different features can be retrieved to illustrate five different *Why feature* dialogues by the example reminding algorithm. The average index in Creanimate includes at least two different values for each type of animal attribute. In other words, an average index in Creanimate contains slightly more than two features, two actions, and two behaviors. Since actions can be used in both *Why action* and *How action* dialogues, this means that the average index can be used to illustrate discussions of at least eight different explanation questions. Table 8 shows the average numbers of attributes that have been associated with the current 206 Creanimate indices. In addition to these attributes which are used for example reminding, seven percent of the indices are associated with expectation-violations, meaning that they can be retrieved by the expectation-violation reminding strategy.

Table 8.–The average number of values in each of the primary slots in the 206 Creanimate indices

Slot Name	**Average number of slot values per index**
Features	2.14
Actions	2.39
Behaviors	2.17
Feafuns	2.10
Plans	2.32

A second way to look at the breadth of coverage of the indices in Creanimate is to look at the number of different objects in the knowledge base that appear in indices. The larger this number, the broader the coverage of the Creanimate story library. Table 9 shows this number both for attributes that appear directly in indices and for those that are linked to one or more indices by inheritance.

Table 9.–The number of objects of each type that appear in indices

Object Class	**Number that appear in indices directly**	**Number that appear in indices by inheritance**	**Total Number in Creanimate**
Animals	111 (32%)	174 (50%)	343
Features	110 (35%)	190 (61%)	310
Actions	121 (48%)	191 (76%)	250
Behaviors	106 (50%)	184 (88%)	210

Table 9 shows that the storyteller can produce a relevant story for between 61 and 88 percent of the attributes of animals that Creanimate knows about. While these statistics show a high coverage rate for the stories in Creanimate, our experience indicates that optimal story coverage for Creanimate's current knowledge base would require approximately twice as many stories as the system currently contains. This degree of coverage would remove the need to start dialogues by suggesting animals and modifications to students and would allow them to choose anything in the knowledge base. The primary obstacle to achieving this goal is not the indexing process, however. It is story availability. Since film libraries do not tend to be encyclopedic, obtaining a wide range of animal video has proved to be difficult.

8.2 The Role of the Indexer

Just as the technology of expert systems gave rise to the new occupation of knowledge engineer, case-based teaching systems have given rise to the occupation of indexer. An indexer is a non-programmer who plays an essential role in developing a case-based teaching system. As the name indicates, an indexer indexes. However, the role of an indexer extends beyond simply labeling stories with indices. In Creanimate, the indexer participates in story selection, video production, knowledge representation, testing, evaluation, and debugging, in addition to indexing. The indexing process has several steps, many of which overlap or alternate.

The first step, story selection, is described in Chapter 3. To assure adequate coverage, the initial story collection stage should generate many more stories than can actually be used in the final system. This gives the indexer sufficient flexibility in revising the story selections as the indexing and knowledge representation progress. The first stage in building a new system once a large enough body of stories has been collected is called pre-indexing. Pre-indexing is a quick and rough way to take stock of the available stories. Raw footage is first previewed and segmented into self-contained stories that illustrate phenomena of interest. Each story is then reviewed for the features, actions, and behaviors that appear in it. In the pre-indexing stage these attributes are only loosely characterized. Thus, instead of identifying different types of claws as *talons*, *pincers*, etc. during pre-indexing they may all just be called *claws*. In the early development of Creanimate, pre-indexing information was recorded in a spreadsheet. Once the pre-indexing information is complete, stories can be grouped together based on shared features, actions, or behaviors. These groups are then used to filter out stories that are not rich enough in their contents or that duplicate each other. Pre-indexing information can also be used to determine which choices of animals and modifications should be presented to students on the program's first screen.

Once the initial filtering of stories based on pre-indexing is completed, the indexer begins the work of knowledge representation and indexing. Knowledge representation and indexing are mutually dependent activities. Starting with the pre-indexing information, the indexer will view a story and identify the number of indices required for it based on the number of different animals of interest in it. He will then begin to enter the indexing information for that story. Indexing is done by successive refinement, so the first time the indexer may only enter what he sees as the most important elements of a story, and he may stick to the level of description used in the pre-indexing stage. In the early stages of indexing, the priorities are getting the indices into the storyteller's library in some form and discovering where the knowledge base needs to be expanded. When the indexer discovers elements in video clips that do not appear in the knowledge representation, he must extend the knowledge representation to include the new concepts before he can index those clips. In this way, the indexing process drives the development of the knowledge base.

Once the indices have been entered into the knowledge base in a rough form, the indexer can begin to test the system's new knowledge by running the program. As he runs the program he can identify indexing errors and additional gaps in the knowledge base. Once the stories have been roughly indexed and partially tested, the indexer refines the indices. Refinements, such as the distinctions between varieties of claws, that the indexer has ignored until this stage, are now added to both the knowledge base and the appropriate indices. To support incidental remindings, abstractions are added to the relationships in indices, and expectation-violations are recorded. The ratings of stories are adjusted to insure that more appealing ones are offered first, and impacts are added to help the storyteller generate accurate introductions.

It is only after the knowledge representation and indices are reasonably well refined that video production begins. Depending on the source of the video, production may involve the editing and splicing of clips and the addition of sound effects, voiceovers, and music, or it may simply involve dubbing from an externally produced source tape. Once the clips have been edited to their final form, they can be copied onto the storage medium that the program will use. In the current version of Creanimate, the storage medium is an optical disk recorded using the standard laserdisc format. The video production process can be extremely time consuming and may require significant technical production skills.

The final stages of indexing focus on testing and debugging. At this stage, natural language generation and understanding receive a great deal of attention. It is important to have as large a population of testers as possible use the system in order to collect as many transcripts as possible. The indexer then analyzes the transcripts for incorrect output and mis-parsed or unparsed input. These errors are corrected by modifying or adding to the information in the knowledge base.

Every stage in the indexing process requires careful thought and great attention to detail. An indexer must have the appropriate temperament for detail work and be conscientious through every step of testing and refinement. While indexing does not require any programming ability, it does require that the indexer understand the ways in which the program will manipulate the information that he is entering into the knowledge base. The ability of the dialogue manager to evaluate students' answers depends on the correctness of the knowledge base. Maintaining the validity of the knowledge base is assisted by a knowledge representation manual (Edelson 1992) that explains the different object types in the system, their slots, and fillers. Because the Creanimate system deals with elementary topics, a layman's understanding of animal adaptation was sufficient background for our indexer. However, a library of reference materials was essential, as was access to our consulting professor of biology.

Once the initial corpus of stories had been indexed, the knowledge base was sufficiently well-developed that indexing became significantly easier. Initially, every index required extensive additions to the knowledge base. During this stage, it could take as long as a day to index one story because the knowledge base required frequent reorganization. Each reorganization, in turn, forced a good deal of re-indexing. However, as the knowledge base grew more and more stable, reorganizations became less frequent and more localized in their impact. In our most recent indexing efforts, we estimated that the indexer was able to add stories at a rate of approximately one hundred a month, excluding video production time.

8.3 The Creanimate Indexing Tool

While the indexer is a vital resource in the development of a case-based teaching system, indexing on a large scale would be impossible without development tools like the Creanimate Indexing Tool. Building a large knowledge base and indexing hundreds of stories is a daunting task. To ease it, we developed a suite of software tools that assist an indexer with the processes of indexing stories, modifying the knowledge base, and maintaining its consistency. Experience in knowledge representation has shown the importance of the ability to view a knowledge representation in several ways. In particular, the ability to view frames both graphically, as a network of links and nodes, and textually, as a set of slots and fillers, is essential. The Creanimate Indexing Tool provides these capabilities as well as giving the indexer the ability to work directly with video clips. The two views on the knowledge base are provided by modules called the *hierarchy browser* and the *frame editor*. Video previewing is provided by the *clip editor*. The Indexing Tool was written in Common Lisp (Steele 1990) on the Apple Macintosh using standard conventions for the Macintosh interface.

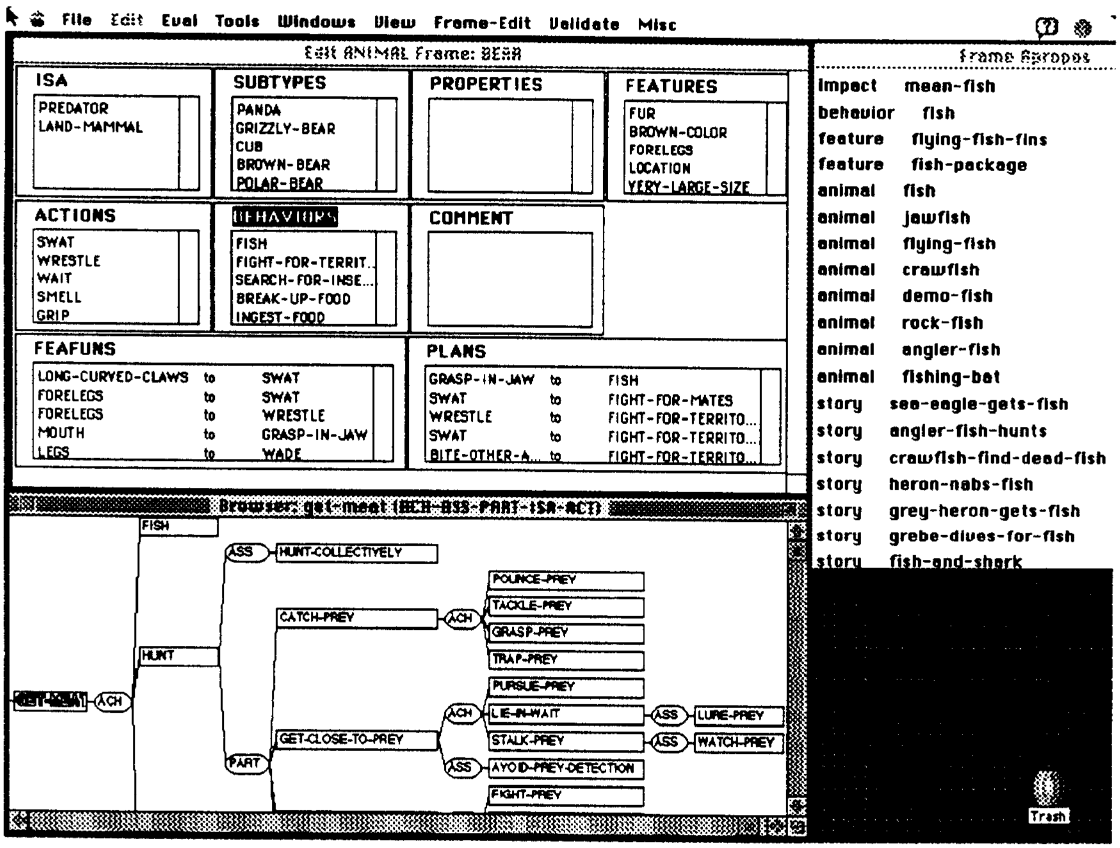

Figure 38. A screendump showing a hierarchy browser at the lower left and a frame editor above it

An indexer views and modifies the knowledge base through the browser and the frame editor. Browser and editor windows can be opened by selecting choices from a menu or by typing commands to a Common Lisp interpreter. A browser window displays a network that shows objects of a single type and the links that connect them. When the indexer wants to view a hierarchy, he opens a browser window by specifying the particular object whose relationships he wants to see. That object is designated the *center node* of the hierarchy displayed in the browser window. Abstractions (parents) of the center node are displayed to the left of the center node, and specializations (children) are displayed to the right. A browser window for the feature *horns* is shown in figure 39.

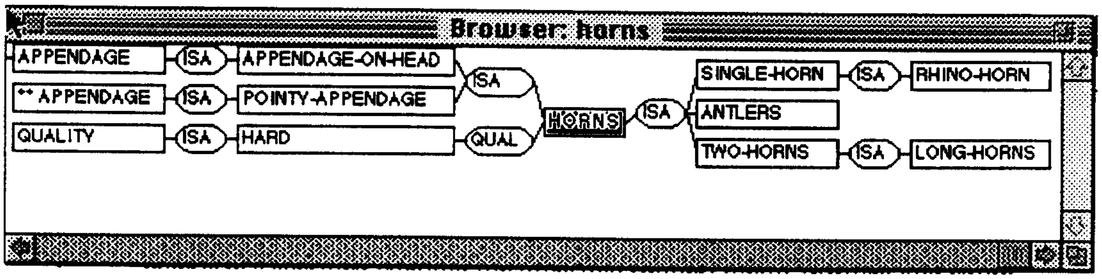

Figure 39. A browser window for the feature *horns*. When the same object appears twice in the same window, it is annotated with two asterisks (**), as with the feature *appendage*.

From within a browser window, an indexer can use the mouse to open a browser window centered on one of the visible objects. He can also choose to view one of the visible objects in a frame editor window. The frame editor provides a very different view of an object. It displays all of the slots and fillers of a single object, as shown in figure 40.

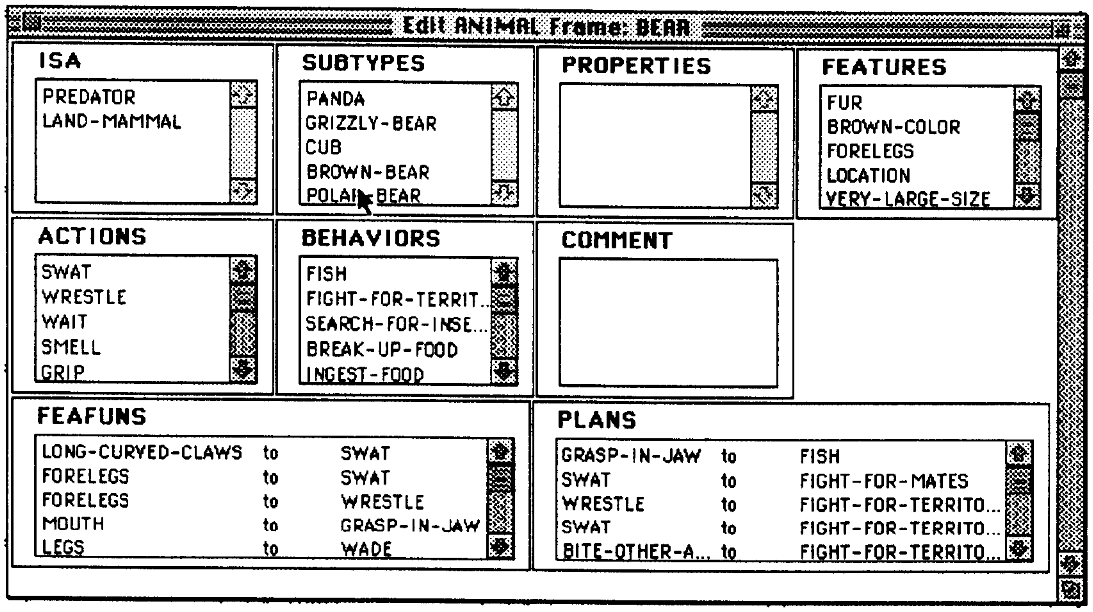

Figure 40. An editor window showing some of the slots in the frame representing the animal *bear*. The remaining slots can be seen by scrolling the window down.

Just as with browser windows, editor windows enable the indexer to use the mouse to open new windows to display any of the visible objects. Editor windows allow an indexer to view a frame in one of two modes: local mode or inherited mode. In local mode, the editor window displays all of the slot values directly associated with the object in the window. In inherited mode, the indexer sees all of the local values plus any values that the object inherits through abstraction links.

Editor windows allow the indexer to modify the knowledge base through the addition or deletion of values from slots. Using the indexing tool, the indexer can modify slots with the standard Macintosh cut, copy, and paste operations. An object is selected by clicking on it with the mouse. It can then be cut or copied with a menu command or keystroke equivalent. That object can then be added to another slot in the currently displayed object or in some other object by selecting the desired slot with the mouse and issuing a paste command. Objects can also be added to slots by clicking on the slot, selecting the "Add Slot Value" command from the menu and then typing in the name of the desired object as shown in Figure 41. The indexer is always asked to confirm any changes, including requests to create new objects, before they are executed.

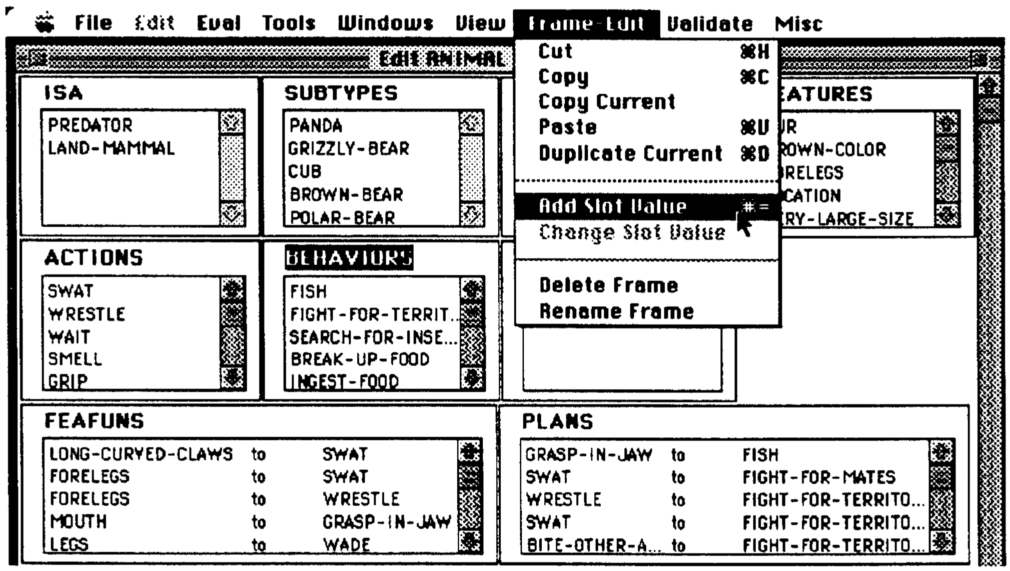

Figure 41. An indexer adding a value to the *behaviors* slot of the animal *bear*. The Indexing Tool will prompt the indexer for the name of the behavior he wants to add. Alternatively, the indexer could select the desired object in a browser or editor window and "paste" it into the slot.

The indexer can also use the frame editor to construct and modify indices. For example, figure 42 shows an index for a story about two octopi fighting over territory.

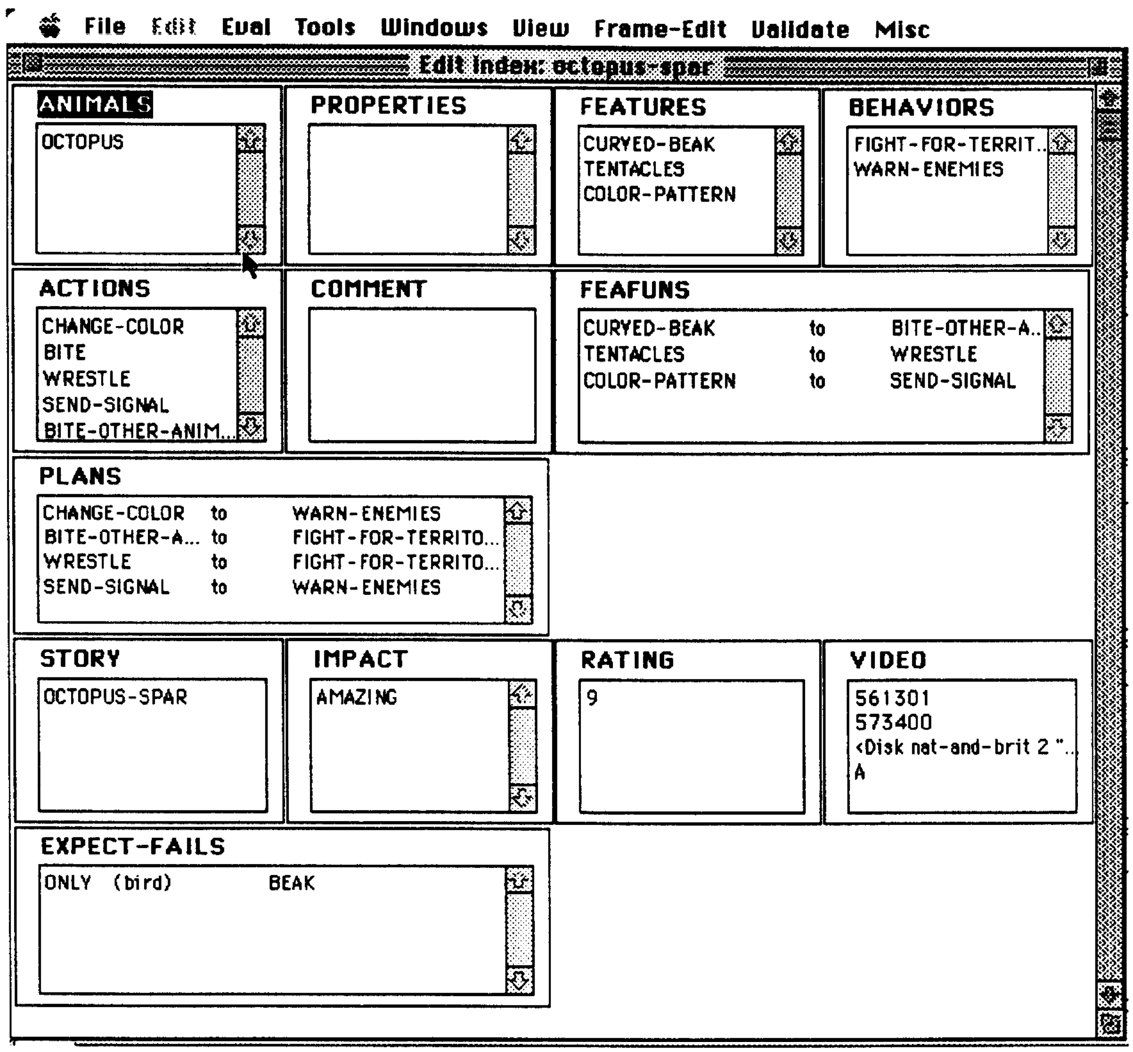

Figure 42. An index for a story about octopi fighting over territory. This index violates the expectation that only birds have beaks. In this story, an octopus uses its beak, which is concealed beneath its body, to attack another octopus.

Locating Objects with the Indexing Tool

Usually, an indexer uses the indexing tool for two one of two purposes. One is to add new stories by previewing video clips, creating indices for them, and adding any new concepts they require to the knowledge base. The second is to debug the knowledge base in response to testing and use of the program. In both of these tasks, locating objects in the knowledge base is central. Consider, for example, indexing a story about Malaysian flying frogs. These frogs have large, webbed feet. They glide from tree to tree by leaping into the air and spreading their toes, allowing the membranes between them to act as wings. In the process of indexing this story, the indexer will need to know if the feature *webbed feet* exists in the knowledge base already. If it doesn't, he will have to add it. He has several options available to him for locating this object. The first option is to request a frame editor window for the frame *webbed feet*. He would do this by selecting "Edit Frame" from the "View" menu and typing in "[feature webbed-feet]". If the frame does not exist, the indexing tool will ask him if he wants to create it. If it does exist, the tool will display it in a frame editor window. However, if he is not certain whether or not the frame already exists, the indexer may prefer not to find the frame this way for two reasons. First, he could easily mistype the name of the frame. Second, the frame might already exist with a different name. For example, he might leave out one of the *b*'s in webbed, or he may have already created with the name *webbed foot* or *web feet.* In any of these cases, he would have accidentally created an unnecessary frame by typing in the wrong name. Errors like these that result in unnecessary duplication are very difficult to identify and correct.

Instead, he might use one of the built-in techniques for locating frames in the knowledge base. The goal of these techniques is to make it quick and easy to locate objects without requiring that the indexer either memorize the names of all of the objects in the knowledge base or be an error-free typist. In many cases, the best way to search for an object is by browsing through hierarchies. To find *webbed feet,* the indexer could call up a hierarchy displaying all of the features in the knowledge base, but he would be more likely to display a hierarchy centered on *feet.* One advantage of searching for objects through hierarchies is that if they do not already exist, the indexer can find the appropriate place to to insert them.

A different way to locate objects is with the "apropos" command. The apropos command allows an indexer to type in a string and see the names of all the objects in the system that have that string in their name. Thus, an indexer could type in either "web" or "feet" and see a list of objects with those substrings in their names. The apropos command is available to the indexer any time he is prompted to type in the name of an object. For example, suppose the indexer wants to add the behavior *avoid predators* to an index, but he can't remember whether it's called avoid-predation, avoid-predator, or avoid-predators. Instead of typing one and hoping he's correct, he can just type "avoid" and press the "apropos" button. The indexing tool will present a list of behaviors with the string "avoid" in their names. The indexer can select the one he wants with the mouse, thereby avoiding errors caused by mistyping. The last way to search for objects is using frame editor windows. If the indexer knew that he had added webbed feet to the animal *duck,* he might call up an editor window for *duck* and look at its *features* slot.

Background Actions in the Indexing Tool

The indexing tool is designed to reduce bookkeeping work on the part of the indexer. For example, without the support of the indexing tool, if an indexer were to add a feafun to an animal, he would also need to check if the the action and feature were present individually in the representation of the animal. If not, he would have to add them. To remove this burden from the indexer, the indexing tool employs *if-added methods.* If-added methods are functions that are associated with individual slots. When an indexer adds a value to a slot, the if-added method for that slot checks to see which additional slots should be updated to maintain consistency. These if-added methods allow the indexer to combine the processes of adding to the knowledge base and indexing stories into one. Since the representation of an animal must possess all the attributes that it displays in a story, adding an attribute to an index automatically results in the addition of the same attribute to the animal. For example, if an indexer added the feafun *raptor's beak in order to tear meat* to a story about an eagle eating a fish, any of the following changes that were necessary would be effected by if-added methods:

Table 10.–The results of adding the feafun *raptor's beak in order to tear meat* to a story about an eagle eating a fish

Value added	To slot	Of object
Raptor's beak in order to tear meat	Feafuns	Index
Raptor's beak	Features	Index
Tear meat	Actions	Index
Raptor's beak in order to tear meat	Feafuns	*eagle*
Raptor's beak	Features	*eagle*
Tear meat	Actions	*eagle*
Raptor's beak	Features	*tear meat*
Tear meat	Actions	*raptor's beak*

In this case, the indexing tool only requires one action where eight would have been necessary otherwise. When the indexer adds or deletes a value of a slot the indexing tool calculates all of the resulting changes and presents them as a group to the indexer for confirmation before effecting any of the changes. This allows the indexer to see all the effects of any change before committing to them.

Safeguards in the Indexing Tool

Maintaining consistency in a large knowledge base can be extremely difficult. The indexing tool contains functions that assist with this process. For example, the frame definitions for the Creanimate objects specify permissible values for every slot. If the indexer tries to add an inappropriate value to a slot, the indexing tool will prevent it. Thus, if the indexer tries to put an animal into the *features* slot of an action, the indexing tool will notify him of his error. In addition, the indexing tool contains safeguards against redundant information in the knowledge base. If the indexer adds a value to an object that the object already possesses, either locally or through inheritance, it will not allow him to add it again.

The safeguards that provide the biggest savings in time and effort are those that search through the knowledge base looking for inconsistencies and missing information. If an indexer only added knowledge to the database and never deleted anything, the if-added methods could be counted on to maintain consistency. However, because the indexer can delete and move objects around, inconsistencies may arise. One inconsistency commonly appears when hierarchies get rearranged so that one object that used to be an abstraction of another no longer is one. The indexing tool contains knowledge validation routines that locate situations like this and brings them to the attention of the indexer. Inconsistencies also arise in hierarchies of objects that are only allowed to have a value in one of a set of slots. For example, a behavior is only allowed to have a value in either its *achieves, assists,* or *part-of* slots. The indexing tool has routines that the indexer can use to search object hierarchies looking for errors like these. Lastly, indexers occasionally leave out required information from objects. All of the features that appear in an action's *features* slot, for example, must also appear in either the *requires* or *suffices* slot of that action. If the indexing tool locates an action with a feature in its *features* slot that doesn't appear in its *requires* or *suffices* slots, then it will request that the indexer add it to one of them.

All of the validation routines that search the knowledge base for inconsistencies and missing information can be run in either batch or one-at-a-time mode. In batch mode, the indexer lets the validation routines run to completion at which point he receives a report of all of the problems they have located. This process can take an hour or more for a large knowledge base. In one-at-a-time mode, a validation routine runs until it finds a problem which it reports to the indexer immediately. After the problem is resolved, the validation routine continues searching from the point where it identified the problem.

Testing in the Indexing Tool

The indexing tool includes facilities for testing the behavior of the program itself. After entering information into the knowledge base or indexing new video clips, an indexer might want to ensure that these objects and stories will appear the way he expects them to in a student's interaction. The indexer can test Creanimate from within the indexing tool in three ways. First, he can run the program itself. Instead of seeing the graphical interface that students use, however, an indexer tests it in a normal text window. In this "command-line" mode, the indexer responds by typing instead of clicking with the mouse. Likewise, instead of actually seeing video stories, he simply sees the title of the story. Figure 43 shows Creanimate running in command-line mode. In command-line mode, an indexer may check to see whether or not a particular story is presented when he expects it to be, and that the dialogue manager is able to use the information in the knowledge base in order to respond properly to a student's input.

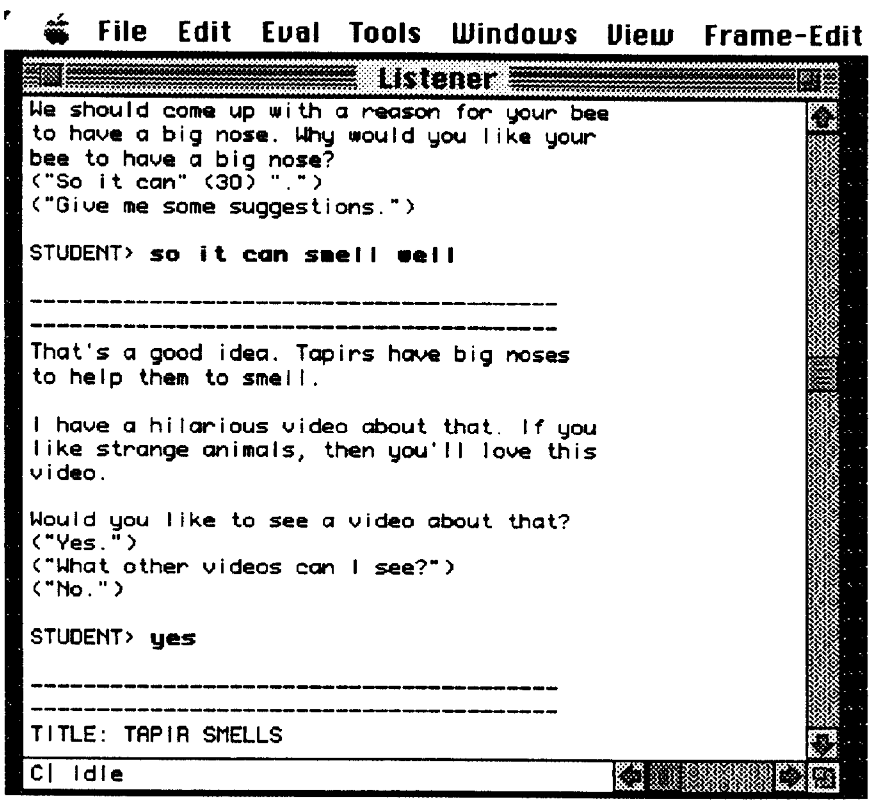

Figure 43. Creanimate being tested by an indexer. The indexer's typed input appears in boldface. This testing facility is always available in the indexing tool.

When the indexer enters new concepts into the knowledge base, he often wants to make sure that the natural language generation and understanding information for those objects operates the way he expects. The indexing tool includes facilities for testing generation and parsing without running the whole program. Figure 44 shows the tool for testing the natural language generator. This tool generates a sentence for every form in which the concept could appear in Creanimate output. The indexer can then review these sentences to verify that the information he has entered is correct.

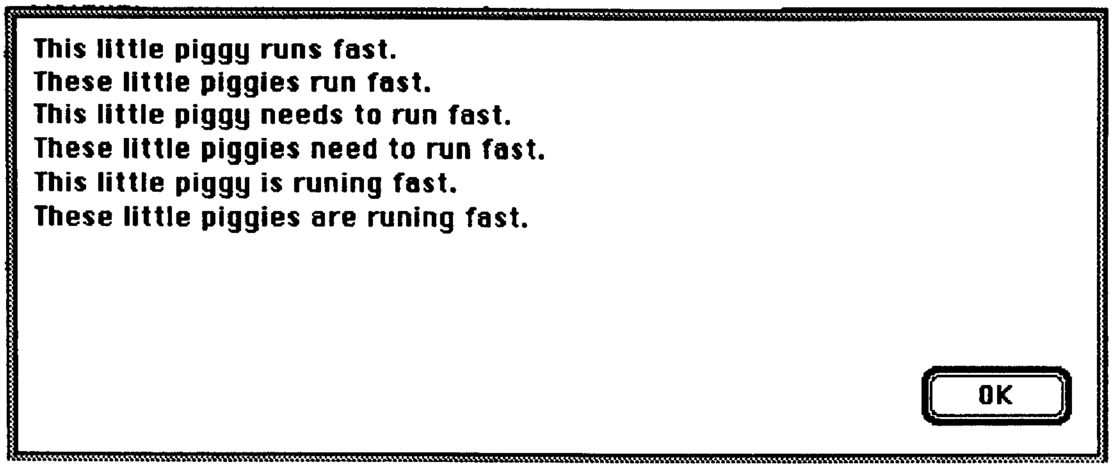

Figure 44. The tool for testing the natural language generator. In this case, the indexer is testing the forms for the action *run fast*. The present participle form, "running" is spelled incorrectly.

A different tool allows the indexer to test the parser. After an indexer adds parsing information to an object, he can test it with the parser tool to make sure that the parser performs the way he expects. Sometimes, the indexer may make a mistake in entering parser information or there may be another concept in the knowledge base whose parsing information conflicts with the new entry. The parser tool helps the indexer to identify these problems. Figure 45 shows the indexer testing to make sure the parser understands the phrase "chase girl bugs" correctly.

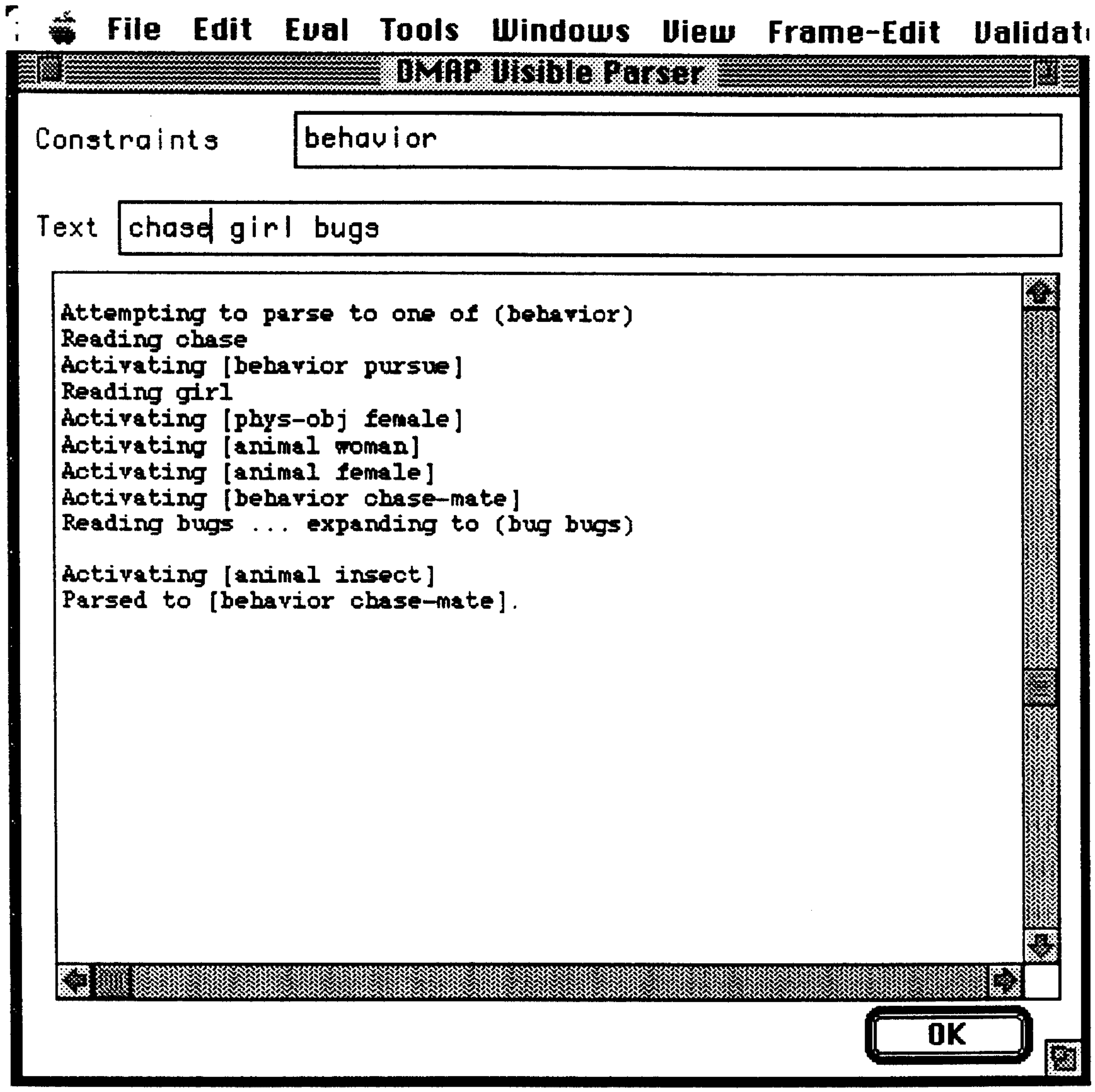

Figure 45. The tool for testing the parser. The indexer enters the type of object that he wants parsed in the "Constraints" box, and the text he wants parsed in the "Text" box. A trace of the parser's actions appears in the main box. The phrase "chase girl bugs" parsed to the behavior *chase mate.*

Working with Video Clips in the Indexing Tool

The indexing tool includes a module called the clip editor, which allows the indexer to view video clips on his monitor as he is indexing them. Using a control panel that looks like the controls for a VCR, the indexer determines the beginning and end of a story. Using a menu command, he can instruct the clip editor to automatically enter the frame numbers of these points into the *video* slot of an index. In addition, the clip editor allows him to select a "preview" picture for the animal in an index. The preview picture is a still picture digitized from the video clip that shows the animal in the index. The preview picture is shown to the student when the storyteller offers him the story. If the animal's name is unfamiliar to the student, the preview picture can be very important in helping him decide whether or not he wants to see the story. The clip editor is shown in figure 46.

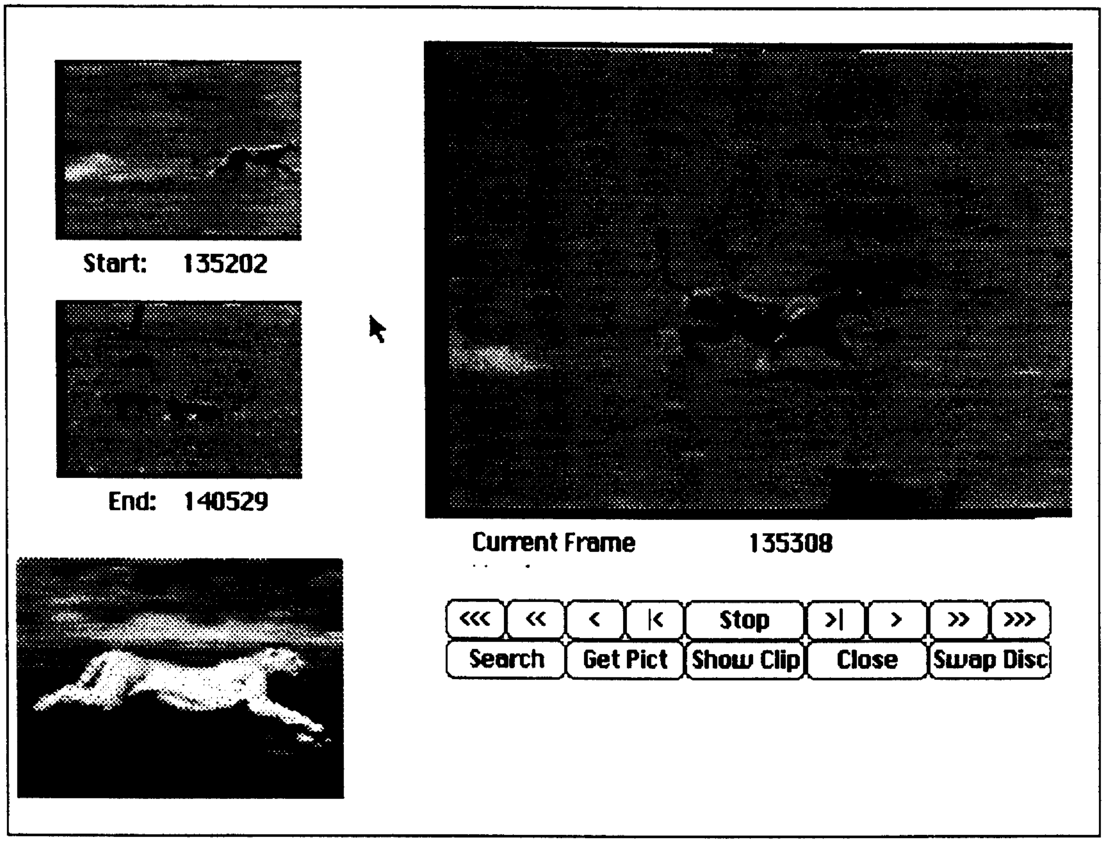

Figure 46. The Creanimate clip editor. The clip editor includes (clockwise from top right), the main preview screen showing live video directly from the videodisk player, the control panel, the digitized preview picture of the animal, the last frame of the clip and the first frame of the clip.

8.4 Summary

Indices draw on objects from the Creanimate knowledge representation in order to describe a story. An index describes the attributes that a particular animal displays in a story. Because indices are directly linked to objects in the knowledge base, the storyteller is able to retrieve them efficiently.

Indexing large numbers of stories and building a large knowledge base is a difficult and complex task. To assist with that process, we have constructed an indexing tool. This indexing tool enables an indexer to view the knowledge base both graphically and textually, and to modify it through a simple interface. It also enables the indexer to maintain the validity and consistency of the knowledge base. Maintaining a knowledge base of significant size can become unmanageable without tools that reduce complexity. The indexing tool was essential in keeping the Creanimate knowledge base and story library manageable as it scaled up from a few hundred objects to a few thousand.

Chapter 9

Perspectives on Creanimate and Case-Based Teaching

In this chapter I place the foregoing discussion of Creanimate and case-based teaching in context. In the first section, I discuss related work in education, artificial intelligence, and multimedia learning environments. In the second section, I describe some ways the research described in this dissertation can be extended in the future.

9.1 Related Work

Three areas of research have a direct bearing on Creanimate and case-based teaching. They are 1) the case method of education, 2) the theory of case-based reasoning, and 3) computer-based learning environments.

9.1.1 The Case Method

The value of teaching with cases has long been appreciated in many quarters. Law schools, business schools, and, increasingly, other professional schools teach with cases. Williams (1991) in a survey of instruction with cases states, "The case method [in legal education] is popular because teachers and students believe that it is an effective way to learn an ill-structured domain, i.e., one that does not have a consistent underlying theory that can act as a structure for organizing knowledge." In the law, the case method of education emerged out of a dissatisfaction with the previous methods of legal education, which were either apprenticeship or lecture (Williams 1991). Apprenticeship was valued for the practicality of its lessons, however an apprenticeship was only as good as the supervising lawyer and as broad as the lawyer's practice. An apprenticeship had the advantage of learning through firsthand experience under supervision, but the quality of that experience and supervision varied greatly from lawyer to lawyer. At the time of the introduction of the case method in the 1870's, the alternative to apprenticeship was private law schools that taught principles of law through lectures. The training provided by these schools was frequently criticized for its lack of practicality. In 1870, Christopher Langdell, Dean of the Harvard Law school introduced the case method for studying the law (Redlich 1914; Reed 1921). It improved on both the potentially uneven coverage of apprenticeship and the remoteness of lectures from practice. Despite significant early criticism, the case method spread widely on the basis of its results, such that today it is the predominant teaching method in U.S. law schools. Because of the success of the case method in training for the law, Edwin Gray, the first Dean of the Harvard Business School, adopted the case method for teaching business too (Copeland 1954).

Similarly, in recent years, medical instruction has come under increasing scrutiny with one result being a shift toward teaching with cases. Traditionally the curriculum of the first two years of medical school has consisted of basic science taught in lecture style. Studies have shown that most of what is learned during this period is forgotten before students begin their clinical rotations in their third year (Gonella et al. 1970; Levine and Foreman 1973). In response, medical schools have been experimenting with a form of case method that Williams (1991) calls problem-based learning. Problem-based medical education is intended to remedy the decontextualization of knowledge, and the passive role of the student that occur in traditional lecture-based medical education.

The case method employed in law, business, medical, and other professional schools is essentially the same. A case that has been selected for its pedagogical value is presented to students. A discussion ensues in which the instructor and the students examine the issues raised by the case. In business and medicine, the case is presented as a problem that the students solve. In law schools, the case is frequently presented as a result and then analyzed to understand its implications.

The case method resembles case-based teaching in its emphasis on learning from concrete cases. However, they differ in the way they employ cases. In the case method, a case is presented to initiate a discussion. The case establishes a context for learning. In case-based teaching, a case is presented in response to a situation in which the case is relevant. The case responds to an opportunity for learning. The two instructional techniques are compatible, though, in spite of their differences. Case-based teaching can be done in the course of a discussion that is initiated by the presentation of a case, in the style of the case method.

While case-based teaching and the case method differ in the way they employ cases, they share important features. They both rely on the presentation of cases to provide situated, contextualized learning. They both emphasize learning interactions that are authentic to the actual use of the lessons learned. Finally, the issues of case selection and case presentation are central for effective instruction by either case method or case-based teaching. As with case-based teaching, the case method requires cases that provide adequate coverage of the subject matter, and present their lessons in a clear and memorable fashion. Employing the case method raises the question of ordering cases, which is handled in case-based teaching by presenting cases in response to students' actions. The order of case presentation in the case method is generally fixed in advance according to a theory of how the subject matter is organized. The order of case presentation in case-based teaching is dynamic, determined by the needs and interests of the student. Thus, while the case method and case-based teaching share many of the advantages of teaching with cases, case-based teaching is able to opportunistically respond to the interests and needs of the student.

9.1.2 Case-Based Reasoning

Just as the case method is case-based teaching's close relative in the field of education, case-based reasoning is its close relative in artificial intelligence. Teaching with cases that respond to context parallels the process of reasoning from cases. In both architectures, features of the current situation are used to retrieve appropriate cases from memory. The primary difference between the two is the way cases are used after they are retrieved. In case-based reasoning the cases are used to explain or resolve the reasoner's current situation, whereas in case-based teaching, they are presented to a student as stories from which to learn. Table 11 shows the similarity between the processes of case-based reasoning and case-based teaching.

Table 11.–Similarities between the processes of case-based reasoning and case-based teaching

Case-based reasoning	Case-based teaching
1. Identify significant features in the current situation in the world. 2. Use features to retrieve case. 3. Adapt case to fit current situation	1. Identify significant features of the student's situation in the task environment. 2. Use features to retrieve story. 3. Construct a bridge to the story from the current situation.

A difficult challenge for case-based reasoning is feature extraction, the process of identifying the significant features of the current situation in the world to use in retrieving cases. This problem is simplified for a case-based teaching system because it deals with a simplified world, the task environment. For example, in Creanimate, the current situation is fully specified at any particular point in a dialogue by the student's current animal, the most recent modification to that animal, the current explanation question, and the answer to that question that is under consideration. Depending on where the student is in the course of a dialogue, different subsets of these features are used by reminding strategies as retrieval cues. In case-based teaching, feature extraction is not the problem that it is in case-based reasoning because each reminding strategy uses a predetermined subset of the current features.

The reminding process in case-based teaching parallels the retrieval process in case-based reasoning exactly. In both architectures, a search algorithm traverses the system's memory of cases comparing the indices it finds to the features of the current situation. In a typical case-based reasoning system, these features are compared to the features recorded in the index according to a similarity metric. The use of this similarity metric is similar to the role played by the abstraction in Creanimate's similarity-based reminding algorithm. In the example reminding algorithm, no similarity metric is necessary because a story must be a precise specialization of the current target concept or the story is not an appropriate example.

Surprisingly, even the indexing based on expectation-violations in Creanimate has precursors in case-based reasoning systems. For example, CHEF (Hammond 1986) indexes the plans in its case library by the failures that they avoid. Since the failure of a plan results in an expectation-violation, indexing by plain failures is the same as indexing according to expectations that are violated. The value of the retrieved cases in both systems is similar, too. Expectation-violations in Creanimate are intended partly to avoid students from maintaining overly strong generalizations. In other words, they help student to avoid mistakes. Retrieving cases by plan failures in CHEF serves a similar purpose. It helps the planner to avoid executing a plan that will fail. Both strategies are intended to avert mistakes.

An important element of the case-based reasoning process that plays no part in the current implementation of case-based teaching is case adaptation. In case-based reasoning, a retrieved case is typically modified according to the differences between the current situation and the case to make the case applicable to the current situation. Since the stories in a case-based teaching system are pre-recorded, they cannot be adapted to fit the current situation. Instead, the adaptation phase in case-based reasoning is replaced with bridge creation in case-based teaching. A bridge is an introduction that communicates to the student the relationship between the student's current situation and the story he is about to see. In future case-based teaching systems, stories may be adaptable, in which case experiences with adaptation in CBR (e.g., Kass 1990) will be valuable.

Overall, the case-based teaching architecture bears a strong resemblance to its cousin, case-based reasoning. They share the same central research issues of indexing and retrieval. As a result, Creanimate has benefited from earlier research in case-based reasoning.

9.1.3 Computer and Multimedia Learning Environments

The use of computers and multimedia in education has a considerable history. It is beyond the scope of this work to summarize that history here. However in this section I do discuss a few topics that relate directly to important aspects of the research described in this dissertation. These topics are intelligent tutoring systems (ITS's), indexed video libraries, and biology by design. As I have mentioned previously, intelligent tutoring systems are a different class of computer-based learning environments that also employ artificial intelligence. Indexed video is a technology that provides students access to large libraries of video through random access. *Biology by design*[1] is an expression that describes several computer programs that, like Creanimate, allow a student to learn about biology through the creation of new animals or ecosystems.

Intelligent Tutoring Systems

While intelligent tutoring systems are similar to case-based teaching systems in their use of theories and technologies from AI, I discuss them here more for their striking differences than their similarities. Nevertheless, Creanimate does follow in the tradition of some work in intelligent teaching systems. For example, two early intelligent teaching systems, SCHOLAR (Carbonell 1970a; Carbonell 1970b) and WHY (Stevens and Collins 1977) used question-and-answer dialogues that resemble those in Creanimate. More recently GUIDON, like Creanimate, modeled its dialogues after Socratic or inquiry teaching (Clancy 1987). GUIDON also used cases as a central part of its strategy for teaching medical diagnosis. However, GUIDON's teaching was an example of the case method not case-based teaching because it selected its cases to establish an instructional context rather than in response to the context. Creanimate resembles another intelligent teaching system in its use of examples. The WEST system (Burton 1982) employed an approach they called *issues and examples,* in which the system responded to errors by students by determining which issues were misunderstood and presenting examples of correct ways to deal with those issues. In a rough sense, the issues of WEST are similar to the explanation questions in Creanimate because they are triggered by student actions and illustrated by examples. A more recent system that teaches with examples is the one under construction by Ashley and Aleven (1992a; 1992b). It retrieves examples from any of five categories to help students learn the appropriate use of precedent cases in legal reasoning.

These systems aside, most ITS research contains fundamental differences in philosophy from case-based teaching. The central issue in most ITS research is student modeling, which is an attempt to keep track of a student's understanding. An ITS typically compares its model of the student's understanding to a model of an expert's understanding and intervenes with appropriate teaching strategies when it sees opportunities to bring the student's understanding closer to the desired, expert understanding. Conceptually, the ITS model has great appeal. It portrays a teacher as a diagnostician who is able to infer the state of a student's knowledge from his or her actions and is able to respond appropriately to correct deficits in a student's understanding. However, the ITS approach has important limitations. First, as I have mentioned in Chapters 1 and 7, it relies on an extremely sophisticated representation of the subject matter that is capable of accommodating models of both novices and experts in the domain. Second, it requires the ability to draw sophisticated inferences about the state of a student's understanding from the limited evidence provided by observing his actions. These

---

[1] John Cleave, a graduate student at ILS, originated the phrase *biology by design.*

limitations currently restrict the applicability of ITS research to task-based, logical domains like algebra, geometry, physics, and computer program. These domains are suitable for ITS's because they lend themselves to rule-based reasoning and because clear goals, plans, and plan-steps can be identified in the tasks within them. However, in more complex domains, those in which there is no strong theory that can be encoded in rules, ITS's will not work. In complex domains, people reason with cases, and the best way to teach students to function in these domains is to expose them to cases that they can use for reasoning.

In contrast, case-based teaching, as an architecture, requires a much less sophisticated ability to represent domain knowledge or the current state of a student's understanding. A case-based teaching system's need for domain expertise is limited to the ability to interpret actions in the task environment with respect to one of its reminding strategies. Its expertise in student modeling and diagnosis is encoded in the reminding strategies themselves.

In addition, through its presentation of cases, case-based teaching supports the natural process of case-based reasoning. In this respect, case-based teaching addresses an issue that has not been well addressed by ITS research. This is the issue that Wenger (1987) calls *knowledge communication*. Significantly, most ITS's are rule-based systems and have at their heart a model of a student as a rule-based reasoner (e.g. GUIDON (Clancy 1987) and WEST (Burton 1982)). The form that their communication with students usually takes is therefore abstract rules. However, as I stated in Chapter 2, psychological evidence indicates that people reason from cases in complex situations. Learning rules will not help with case-based reasoning. On the other hand, learning from firsthand experiences and stories does provide cases to support case-based reasoning. In contrast to the rule-based knowledge communication employed by ITS's, a case-based teaching system uses stories to provide students with the cases that they can use in the natural process of case-based reasoning.

One final difference between the ITS approach and case-based teaching is what they choose to model. An ITS models knowledge whereas a case-based teaching system models interest. An ITS maintains a model of a student's current knowledge and acts according to what it believes a student knows and doesn't know. The case-based teaching architecture could employ such a model in the decision to tell a story. However, case-based teaching systems are more concerned with what a student may or may not be *interested* in at a particular moment than what he may or may not know. One of the problems facing ITS's is that very often a student acts as if he understands a concept at one moment and then acts as if he doesn't understand it shortly thereafter. It can be difficult for the ITS to maintain a model of the student's knowledge in these circumstances, and therefore hard to determine when and how to intervene.

The problem is that people do not categorically know something or not know it. Sometimes they only know things partially, or they forget them temporarily, or they lose confidence in what they thought they knew. This is not a problem for a case-based teaching system which monitors a student's situation looking for times when he might be interested in a story as opposed to times when he *needs* a story because he lacks some knowledge. People are interested in stories that confirm their knowledge as well as stories that correct their ignorance. A case-based teaching system contains a model of when a story is relevant based on the context that a student is in, not based on a model of what he knows. It is relatively easy to predict what a student will be interested in seeing when you know what decisions he is facing. It is significantly harder to predict what a student knows based on what actions he has taken. For this reason, maintaining a model of a student's interest is much more valuable to a case-based teaching system than a model of his knowledge.

Indexed Video

One of the most exciting educational technologies to become available in recent years has been random-access storage of motion picture footage. Advances in technologies such as videodisks, CD-I, and digital video compression make it possible for a viewer to instantly access single video clips from increasingly large video libraries. Instructional designers have long recognized the power of motion pictures as a communication medium and are rapidly capitalizing on the new random-access technologies. The instructional products based on these technologies usually fall into three categories. The simplest category does not use a computer, but provides a user with a well-organized listing of the contents of a video library and an interface to the technology that allows the user to immediately view any clip that he or she chooses. This interface may consist of a remote control that allows the user to type in the starting frame number of the clip or a bar code reader that translates a bar code into the frame address of the video clip. Systems like these are particularly useful as aids to teachers in the presentation of lectures. Teachers can prepare material in advance and then illustrate a lecture with relevant film clips.

The second category of indexed video system incorporates a computer interface. These systems put the capabilities of computer database and hypermedia technologies at the disposal of the user. A user might be able to locate a clip he is interested in by doing a keyword search through a database of descriptions of video clips or by browsing through a hypermedia system. Such systems enable a motivated user to locate video clips of interest either by searching for them using direct queries or by browsing in a "discovery learning" mode. The number of indexed video systems with computer front-ends is already enormous, and several commercial educational products are devoted to the same subject matter as Creanimate. One, *The Encyclopedia of Animals* distributed by Optical Data Corporation (Optical Data Corporation 1989), consists of eight one-hour videodisks containing clips that similar to those employed in Creanimate. These videodisks come with a Hypercard stack that allows a user to search for clips by topic or word, follow pre-recorded "tours" through the clips on a disk, or to browse the contents of a disk directly. Another, *Mammals: A Multimedia Encyclopedia* is an award-winning CD-ROM produced by the National Geographic Society (National Geographic Society 1989). While *Mammals* only contains 45 actual video clips, it includes 700 color photos and 155 animal vocalizations. The interface lets children search for animals by name or browse by family.

The third form of indexed video system is the case-based teaching architecture described here. The presence of the task environment is a key difference between case-based teaching and the other indexed video architectures described above. The search and browsing interfaces described above rely either on a motivated learner who knows what he or she is looking for on the value of discovery learning. However, in the absence of a task, a student might have no motivation to view the video clips that are available to him and no reason to pay attention to any of their contents. A task environment with a motivating task provides a student with a reason to view to video clips, and a context through which to interpret them and index them in his memory. Several other case-based teaching systems in addition to Creanimate are under development at the Institute for the Learning Sciences.

Biology by Design

Creanimate uses a task in which a student studies something by constructing it. This "constructionist" (Papert 1986) approach is not inherent to case-based teaching systems but is an effective means for achieving active learning. When a student creates something himself he becomes invested in it in a way that provides a motivation for him to learn principles that apply to it. In addition, designing an artifact provides valuable opportunities for learning. In the course of refining a design for an animal, for example, a student can confront the same issues that need to be explained when considering an existing animal.

Several other computer-based systems besides Creanimate use this approach to teach about animal biology. Designasaurus II, a commercial program produced by the Britannica Software (Patterson 1990), gives the user the opportunity to create a dinosaur by selecting various body parts from other dinosaurs. The user then places his creation in one of several prehistoric ages and climates, where he controls it while it searches for food and water, avoids predators, and looks for mates in a simulated environment. The simulated environment provides a video game in which the user gains points by helping his dinosaur to survive. Because the physical features of the user's dinosaur determine what his requirements for food and rest are as well as what strategies he can pursue to locate food and avoid predators, the program can help a user to learn the relationships between features, activities, and behaviors. However, because the program is designed more for entertainment than for education, many opportunities to teach or reinforce lessons are missed.

A second commercial product also employs a simulation of animals in their environment. SimLife, produced by Maxis (Karakotsios 1992), simulates an ecological system composed of animal and plant species specified by the user. The user can either create his own plants and animals or use ones provided by the system. A user invents an animal species by specifying parameters that determine its food sources, its means of locomotion, its defenses, its reproductive rate, its mutation rate, etc. The user also controls the environment by choosing the terrain, climate, and initial distribution of plants and animals. Once the simulation starts, the user can observe individual animals acting and interacting, as they move, feed, mate, reproduce, and die. The simulation can be observed through a display of the animals and plants showing their locations in the environment and through graphical displays that show population distributions and trends. The passage of time within the simulation can be controlled by the user so that he can watch things change over hours, days, seasons, or years. The user is free to establish his own goals in using the simulation; he may try to construct a stable population or he may want to observe how quickly different populations grow or perish. One of the difficulties of learning from SimLife, as with many complex simulations, is identifying the causes for the phenomena observed. Trends develop over time, but if time is passing too fast then it can be impossible to identify causes. On the other hand, if time is passing too slowly trends are not perceivable. Even if the passage of time in the simulation is at the right scale to observe a particular trend, the causes of the trend may be difficult to distinguish among the complexity of possible interactions. Nevertheless, SimLife provides an effective and dramatic window into population dynamics, evolution through genetic mutation, and the interactions between attributes of animals and their ability to survive in various environments.

On a different front, the MIT Media Lab has produced several strands of systems designed to teach biology by design. One is the Vivarium project, conceived by Alan Kay and pursued in concert with Apple Computer. The ambitious goal of the Vivarium project was to allow students to create vivid, realistic simulations of complex animals and their environments. While that goal has never been achieved, they have produced several, individual components of their desired system. For example, BrainWorks (Travers 1988), allows a user to construct a simple nervous system for a Logo turtle[2] and then observe that turtle interacting with other turtles and its environment. AGAR (Travers 1988) extends the capabilities of animals in BrainWorks by increasing their possible range of behaviors, giving

---

[2] Logo is a simple programming language designed to help children learn mathematical and geometric constructs through programming. It is described in Papert (1980).

them the ability to activate scripts and allowing them to learn. Petworld (Coderre 1988) is a third system under the umbrella of the Vivarium project. Petworld simulates animals through a set of behaviors that include fighting, foraging, eating, building nests, and exploring. Apparently, none of these particular systems was used with children. Instead, they were considered steps toward a system that children could use to learn about animals by constructing them and their environments and observing simulations.

Other biology by design systems developed at the Media Lab, however, have been used with students. LEGO/Logo (Resnick 1988) allows students to build "artificial" animals that move around in the real world. These artificial animals are constructed out of LEGO blocks with motors, touch sensors, and light sensors. Students write control programs for their creations using the Logo programming language. The students are encouraged to consider the parallels between their machines and how animals and people interact with their environments. It also helps them to see how the complex behaviors of animals can be built out of simple processes.

*Logo (Resnick 1991a; Resnick 1991b) is another Media Lab system designed to teach biology by design. *Logo (pronounced "star-logo") is a set of extensions to Logo that allows a student to program large numbers of Logo turtles that all coexist in the same environment. *Logo allows the student to program both the characteristics of a turtle and characteristics of the environment. In one example provided by Resnick (1991a), a hypothetical student programmed ants that could locate their food and their nest by following pheromones left by other ants. *Logo enables a student to populate a world with large numbers of his animals, each of which operates according to the same simple algorithm. An important lesson provided by observing *Logo scenarios is that the animals' behaviors seem to indicate a central control mechanism. When people looked at the ant system described they had the strong impression there must be an organized, central controller instructing the ants on how to collect food and bring it to the nest. However, this appearance of centralized control emerges from the individual interactions among the ants, their environment, and each other. This is a valuable lesson in understanding the complex activities and constructions produced by societies of very simple organisms in the natural world.

To summarize, Creanimate is one of a number of computer-based environments that teach biology by design. They each use different techniques to teach different lessons. Some, such as SimLife and *Logo, are focused on interactions among large numbers of animals and their environment. Others, e.g., BrainWorks, AGAR, and LEGO/Logo, focus on the relationship between an animal's control structure or nervous system and its external behaviors. Finally, Creanimate and Designasaurus focus on the adaptability of an individual animal, in particular the relationships between its physical features and its actions and survival behaviors. Underlying all of these systems is the belief that constructing animals is a valuable way to get students invested and interested in learning about animals and how they survive. Interestingly, each of the other systems is compatible with the case-based teaching approach embodied by Creanimate. Any one of them could be coupled with a storyteller that would help students to learn from their experiences and observations of their creations. Both the general case-based teaching architecture and the Socratic case-based teaching architecture of Creanimate could be useful adjuncts to these biology by design systems. Using the general case-based teaching architecture, the storyteller would observe the student's experiences with his or her creations and present stories when they are relevant. Using the Socratic case-based teaching architecture of Creanimate, a dialogue manager could respond to students' situations with questions which would establish opportunities for learning from stories.

9.2 Future Directions

We move now from the topic of related works to a discussion of how the research described in this dissertation can be extended in the future. In Chapter 1, I outlined a set of ambitious goals for developing computer-based learning environments that addressed the concerns of researchers in education and computer science, educators, parents, and children. The work described in this dissertation is one step toward achieving those goals. In looking forward to the next steps in this research program, there are two important questions: 1) How should Creanimate itself be extended? 2) How can the lessons of Creanimate be applied in order to further the development of case-based teaching?

9.2.1 Extending Creanimate

The Creanimate system is an experiment designed to investigate reminding in support of teaching. In developing Creanimate, our goal has been to develop and implement a theory of reminding based on theories of the roles of stories in learning and cases in understanding. The resulting architecture establishes a context for teaching with stories through a Socratic-style dialogue in which the computer responds to students' hypotheses with thought-provoking questions. It capitalizes on the resulting opportunities for learning by presenting stories retrieved by three different reminding strategies. Creanimate, as an implementation of this Socratic case-based teaching architecture, has demonstrated the potential of this approach. In extended sessions, it conducted natural question-and-answer dialogues with elementary school students and presented relevant stories at appropriate moments. This preliminary success notwithstanding, Creanimate has great room for improvement. Some of these potential improvements contain lessons for case-based teaching in general, and some are particular to Creanimate.

Evaluating Educational Effectiveness

A great deal more study needs to be done before sound conclusions about the effectiveness of either Creanimate or case-based teaching can be made. In the course of developing Creanimate, we also created a set of assessments designed to determine the effectiveness of Creanimate. These assessments presented students with pictures of unfamiliar animals and asked them to either write down their questions about those animals or to answer explanation questions about them. The goal was to determine whether using Creanimate had an effect on the sorts of questions that students and explanations students generate for animals. In particular, the hypothesis we were hoping to verify was that as a result of using Creanimate students would ask more explanation questions and use case-based reasoning more often in constructing explanations. Unfortunately, our preliminary testing revealed no statistically significant shift in question-asking or explanation. We attribute these results to the relatively short period of time that students used the program distributed over several weeks (approximately forty-five minutes a week for three weeks).

The point of mentioning these evaluations here is not to claim educational effectiveness for Creanimate. Instead, the point is to indicate the direction that such evaluations should take in the future. The focus of this work, up to this point has been developing and implementing a theory in artificial intelligence. The theory describes the role of reminding in teaching, and the architecture is the Socratic case-based teaching architecture embodied by Creanimate. In the next stage of this research program, this architecture should be evaluated to determine whether it is effective in achieving the goals stated in the theory. However, the evaluation methods must be consistent with the philosophies of learning and understanding that underlie the architecture. A traditional approach to assessment would have tested students' commands of facts about animals in the wild before and after the use of the program. These results might have been compared to those of students engaged in some control activity, such as reading a textbook, hearing a lecture or viewing a film. However, Creanimate is not designed to improve a student's recall of facts, and in spite of any incidental improvement it might cause, it should not be evaluated that way. Therefore, one important way to extend the research begun with the Creanimate system is to continue to explore assessment techniques that are suited to the objectives of case-based teaching[3].

Any studies of the educational effectiveness of computer-based learning environments should also be concerned with the influences of using the systems on attitudes and motivation. In our preliminary evaluations of Creanimate we interviewed the students to determine how they viewed Creanimate in comparison to standard school activities and also to leisure activities such as watching TV or using video games. These evaluations are important because one goal of this research program is to counter the negative perceptions of science that students currently develop in our school systems. The Creanimate system is an attempt to achieve this goal by producing an environment that children enjoy, and that sparks their curiosity about science. As I described in Chapter 1, students responded extremely favorably to Creanimate. They became deeply absorbed in their sessions, were reluctant to stop using it, and eager to return for successive sessions. In the surveys we conducted, they expressed a strong enjoyment of the program. Success in achieving this goal with individual systems such as Creanimate, will help to improve students' attitudes toward science in general. To document these changes, it will be important to continue to monitor students' enjoyment of Creanimate and the program's influence on their attitudes about science and science learning.

Speed and Memory Issues

In building educational systems, even experimental ones, the issue of hardware affordability must be confronted. School systems have, and will continue to have, only limited resources to expend on computer technology. While Creanimate was never intended to be a system that could be deployed on currently affordable platforms for schools, we wanted to show that it is feasible on hardware that would be affordable within the next five years. With the support of IBM, we developed a version of Creanimate that runs on an IBM PS/2 Model 80[4]. This was a difficult task because the size of Creanimate's knowledge base strained the computer's memory capacity, and the program's reliance on search in order to draw inferences and retrieve stories strained the computer's processing speed. In fact, while we were able to demonstrate the feasibility of deploying systems like Creanimate on affordable hardware in the near future, we have not been satisfied with the speed at which it ran on this hardware. Typically, it required more than a second between steps in a dialogue, and pauses of up to nine seconds occasionally occurred while the dialogue manager evaluated a student's typed response or the storyteller searched for remindings. (For comparison, running on an IBM RS/6000 UNIX workstation, the response time of Creanimate was never longer than 1/2 second and was typically less than 1/10 of a second.)

---

[3] To avoid any possible misunderstanding, I feel it important to mention that the only reason I am advocating evaluation and assessment is to document the educational effectiveness of case-based teaching systems, not to provide evaluations of the progress of individual students.

[4] The cpu in this model is an Intel 80386 running at a clock speed of 25 MHz. Running Creanimate required that the computer be configured with a total of 16Mbytes of RAM.

Any honest appraisal would have to conclude that in the current implementation of Creanimate, the slow response of the computer interferes with the program's educational effectiveness. Because students had to wait for the system to respond, their attention would occasionally wander. In addition, students rarely backed up in their dialogues, changed their answers, or chose to see what other videos were available. Our hypothesis is that the slow response of the system made them feel that they had a large investment in time in getting to their current point so they were not inclined to experiment or change their minds. While the undeniable ill effects of Creanimate's slow speed were discouraging, the generation of hardware on which Creanimate was implemented has already been replaced by faster, more powerful computers. Because Creanimate was written in Common Lisp using CLIM, the Common Lisp Interface Manager (Symbolics Inc. 1991), a machine-independent interface language, it can be run on any other machine that supports these widespread implementation languages.

There are two conclusions to be drawn from this experience with trying to implement an experimental AI system on a low-end computer. First, the speed at which an educational system runs does have an important impact on its effectiveness. Despite the fact that case-based teaching systems are AI systems that rely on search through large knowledge bases, they must be designed to run efficiently on affordable hardware. If, as they grow more sophisticated and more powerful, they continue to remain just above the capabilities of affordable hardware, AI-based learning environments will never have an opportunity to make an impact. In Creanimate, slow speed could be disregarded as an implementation detail that could be dealt with later. However, it cannot continue to be treated that way in future systems. It must be treated as the important issue that it really is. Second, and more positive, while it is not possible to claim that knowledge-intensive case-based teaching systems can be deployed on hardware that is widely available in schools today, current trends make it reasonable to believe that the hardware to support these types of systems will be affordable for schools in the foreseeable future. To conclude, in the short-term it is acceptable for experimental systems to be impractical in their current market. However, in the longer term making experimental systems both practical and affordable must become an important part of the experiment.

Limitations in Representation

In reviewing transcripts of students using Creanimate, the most obvious limitation of Creanimate is the inflexibility of its knowledge. Every once in a while, Creanimate lives up to the stereotype of the computer as being overly literal and logical. The difference between the flexibility of Creanimate and its ability to use its knowledge and that of a person shows up in the following dialogue.

```
Student: A Frog that can hunt.
One part of hunting is getting close to its prey.
How would you like your frog to get close to its prey?
Student: By chasing after them.
I don't know any animals that pursue to get close to their
prey, but I do know about some reasons that animals do
pursue.
Would you like to see reasons to pursue or would you like
talk about how you want your frog to get close to its prey?
Student: Stick with pursuing
```

```
White breasted sea eagles pursue. White breasted sea eagles
pursue their prey.
I think this video is great.
Is that something you would like to see?
```

To the student, the behavior of Creanimate in this transcript would look very curious. The program is saying that chasing after your prey is not a way to get close to your prey, but it is a way to pursue your prey. What's the difference? The problem here is that at the time this transcript was recorded, Creanimate's representation of the behavior *hunt* decomposed it into four mutually exclusive sub-behaviors. Two of them were *get close to prey* and *catch prey*. (The other two were *detect prey* and *subdue prey*.) Getting close to prey is performed by stalking or lying in wait. The student's answer "Chasing after them" was interpreted as *pursue*, whose specialization, *pursue prey*, is part of *catch prey*. In other words, the student answered with a way to *catch prey* while the computer was looking for ways to *get close to prey*. People generally don't maintain these sorts of strict distinctions in their thinking about behaviors, however Creanimate's representation requires that distinctions like these be made. This particular problem in the representation of *hunt* has been fixed in more recent versions of Creanimate, but the overall problem, the rigidity of the representation, has not yet been solved.

To a certain extent, Creanimate's inability to handle situations like these more flexibly can be ascribed to the approach of minimal representation described in Chapter 7. It was an important priority in this research to keep the representation as simple as possible in order to focus the research on issues of indexing, reminding, and dialogue management. As a result, the representation could be made more sophisticated and more powerful to handle situations like the one above. In continuing to develop Creanimate, it will be important to increase the expressiveness and sophistication of the knowledge representation. It is the biggest limiting factor for the abilities of the dialogue manager. Therefore, it should be enhanced to cover new sorts of concepts like environments, interference among features, and interactions among animals as I described in Chapter 7. In addition, it should be enhanced to include more refinement in its understanding of the concepts it can already represent. However, we must also recognize the inherent limitations in this approach. Knowledge expressed using traditional symbolic AI techniques, such as Creanimate's semantic network, is rigid. Inference through deductive retrieval can not match the flexibility of human reasoning. While the ability of Creanimate's dialogue manager to participate in a natural dialogue can be significantly improved through successive refinements of its knowledge representation scheme, it would be naive to say that it would approach the naturalness of human discourse. Achieving this goal will require significant breakthroughs in knowledge representation, not evolutionary refinements.

Additional Reminding Strategies

The reminding strategies employed by the storyteller in Creanimate do not exhaust the possibilities for a case-based teaching system. Several additional incidental reminding strategies for Creanimate have already been identified. These include strategies for retrieving extreme remindings and opposite remindings. Extreme remindings will respond to a student's request for a particular modification to an animal by showing the extreme forms of that modification in nature. Thus, if a student asked for an animal that could run fast, an extreme reminding would present the fastest running animal. Opposite remindings also respond to a student's changes to his animal. Opposite remindings show stories about animals that have an attribute that is somehow opposite to an attribute of the student's animal. For example, the opposite reminding strategy would respond to a student's request for an animal that runs fast with a story about an animal that is notable for being slow-moving. Two unanswered questions in this research are how many different reminding strategies can be identified for Creanimate, and how well do they generalize to other domains and task environments. The example reminding strategies clearly generalize to any system that employs the sort of explanation question-based dialogues that Creanimate uses, as do the similarity and expectation-violation reminding strategies. These strategies all generalize because they do not hinge in any way on the domain of animal adaptation. In fact, they are appropriate for any dialogue that deals with how and why questions about three-way relationships. On the other hand, the opposite and extreme remindings described here would probably generalize to other discussions about design (i.e. relationships between form and function) but not necessarily to discussions of other types of explanation questions.

9.2.2 Beyond Creanimate

As we look to the future, it is important to think not just about how to extend and improve Creanimate, but how to take the ideas that were developed through Creanimate and apply them elsewhere. Creanimate has been a valuable platform for testing and extending the theory of case-based teaching. While it was constructed to teach in a specific domain, many aspects of its design are general and have wide applicability. In Chapter 3, I argued for the general usefulness of the case-based teaching architecture. A clever designer can invent engaging task environments for virtually any domain. Using the guidelines for constructing storytellers presented in Chapter 3, one can build storytellers to accompany these task environments. However, before case-based teaching systems can be produced in large numbers, these guidelines must be encoded in tools that support the production of new systems. The Indexing Tool described in Chapter 8 is one such tool. While it was designed specifically for Creanimate, it is, in fact, a general-purpose tool that can be used for building knowledge representations and indexing stories in any domain. It is specialized for the Socratic case-based teaching architecture in that it indexes stories according to questions and provides support for the similarity-based and expectation-violation reminding strategies. Future research in case-based teaching systems should provide insights on how to further assist and automate the construction of task environments and storytellers. To understand what these tools will do and how they should look, it will be important to take the ideas developed through Creanimate and apply them in new domains. This experience will provide important knowledge to inform tool development. In this section, I focus on three widely applicable aspects of the Creanimate system.

- The reminding strategies.
- Socratic-style dialogues.
- Design tasks.

By applying these ideas in case-based teaching systems in diverse subject areas, we will gain valuable knowledge about the general effectiveness of case-based teaching and about the process of constructing case-based teaching systems. This knowledge can in turn be used to build more effective teaching systems more easily.

The Reminding Strategies

Because the reminding strategies employed in Creanimate are not specific to the domain of animal adaptation, they can be employed in a broad range of Socratic case-based teaching systems. To apply these strategies to new domains, it will be necessary to develop an appropriate knowledge representation for the target domain. This representation will serve as the indexing vocabulary. As with Creanimate, that representation must rest on the fundamental relationships that link the concepts in the domain. Once these relationships have been identified, then particular stories can be indexed according to the relationships they exemplify. Examples of these relationships can be retrieved by the same example reminding algorithms used in Creanimate. Similarity reminding can be performed using the same search algorithm that is used in Creanimate. The similarity metric can also be borrowed from one of the two approaches employed in Creanimate or a new one that is more appropriate to the domain could be identified. In addition, standard expectations could be expressed through all-rules, no-rules, and only-rules in order to produce expectation-violation remindings at appropriate moments.

The reminding strategies in Creanimate were designed to be applicable to dialogues about explanation questions in general, not animal adaptation in particular. Through the development of new Socratic case-based teaching systems, we will be able to explore the range of applicability of these reminding strategies. In addition, we will likely discover new reminding strategies with different ranges of applicability. One long-term goal of this research program should be to increase the number of available reminding strategies and to develop a characterization of which sorts of reminding strategies are appropriate for which types of task environments and which range of domains.

The Socratic Dialogue Management Tool

The dialogue manager in Creanimate is Socratic in the sense that it is a responsive questioner. When the student proposes an animal, it responds with a thought-provoking question about that animal. In addition, like a Socratic teacher, it pursues follow-up questions to examine the ramifications of students' answers. Teaching with questions in a case-based teaching system is designed to achieve two important pedagogical goals. It teaches students the explanation questions to ask when confronted with novel situations, and it establishes a context for storytelling. Questions both motivate students to attend to stories and provide them with a valuable structure for indexing the stories as cases in their own memories. Since the value of responsive questioning extends far beyond the domain of Creanimate, the Socratic case-based teaching architecture explored by Creanimate will have value for a large range of subjects.

One way this research can be extended beyond Creanimate, then, is by developing a Socratic Dialogue Management Tool. This tool will allow a system designer to generate new dialogue plans by simply describing the explanation questions for the domain in an appropriate representation. This representation will resemble the question data structures described in Chapter 6. With this information the Socratic Dialogue Management Tool will be able to automatically generate dialogue plans for the explanation questions provided by the designer. This tool will rapidly increase the rate at which new dialogue plans can be added to Socratic case-based teaching systems.

On a similar front, Creanimate's dialogue manager does not make use of all of the questioning strategies that have been associated with Socratic teaching. Stevens & Collins (1982) have identified a number of different strategies that inquiry teachers pursue. In the GUIDON system, Clancy (1987) included many of these strategies for helping students to recognize their misconceptions and increase their understanding. Another way to build on the groundwork laid by Creanimate will be to increase the repertoire of questioning styles pursued by the dialogue manager in subsequent Socratic case-based teaching systems.

Design as a Task for Case-Based Teaching

One aspect of Creanimate's task environment is not inherent to case-based teaching but raises interesting prospects for future case-based teaching systems. That is the aspect that I described I called biology by design in Section 9.1. Learning about something by designing an artifact has several features that are important for learning. First, students become invested in their design providing them with a valuable motivation for learning. Second, the particular features of their design give a student a perspective from which to learn, because information can be interpreted in terms of how it relates to the student's design. Third, designing something gives a student a meaningful goal, which is another important source of motivation and provides a student with a yardstick by which to measure his progress.

One of the insights of all the biology by design systems I described earlier is the recognition that a phenomenon that is not usually considered to be the result of a design process can still be studied through design. Thus, we have systems that teach about animals and ecosystems by allowing children to create their own. In the process they learn about the principles and processes that affect animals and ecosystems in the natural world. It is an obvious extension of the research described here to build systems that teach about design through the act of design. For example, case-based teaching systems that teach automotive or aeronautic engineering can be built along exactly the same lines as Creanimate. This will be a valuable way to extend the research described here. However, a less obvious extension, but one that may have significant value, will be to identify other domains that, like animal adaptation, can be studied through design even though they are not ordinarily considered design domains. So, for example, a student could study political science by designing a government. A Socratic dialogue manager could ask the student about the implications of his decisions, and a storyteller could present cases of governments from history. In another area, a student could study comparative religions by designing his own religion. In response, a case-based teaching system could present relevant examples from the history of religion. The student could learn about the ramifications of his decisions and modify them accordingly.

The paradigm of learning through design has great potential for the case-based teaching architecture because the design process provides the engagement and challenge necessary for an effective task environment. The design task also gives students the motivation, perspective, and structure to allow them to learn from stories and develop a well-indexed library of cases for future use.

9.3 Conclusion

Creanimate has been a platform for developing and extending a theory of reminding in teaching. It demonstrates the capabilities of a case-based teaching system to establish a context for learning through an engaging task environment and to capitalize on that context with stories. Furthermore, it embodies the Socratic case-based teaching architecture which uses a dialogue based on responsive questioning as the means for establishing valuable opportunities for learning. Through this research, we have developed an architecture that rests on theories of learning and understanding that have emerged from research in artificial intelligence and cognitive science. This case-based teaching architecture provides the cases that students require in order to be effective case-based reasoners.

The challenge of constructing case-based teaching systems is to build engaging task environments and to design effective storytellers that are able to respond to the situations that arise in the task environments with appropriate stories. In particular, a storyteller requires a library of stories indexed according to the situations in which they are relevant, a means of communicating with the task environment, and a set of reminding strategies that enable it to locate appropriate stories when they become relevant. In the course of this research, we have identified and implemented three categories of reminding strategies that are useful in Socratic case-based teaching: example remindings, similarity-based remindings, and expectation-violation remindings. Each of these serves a particular set of pedagogical goals, requires a certain type of information to be present in indices, and is appropriate for a particular set of situations in the task environment.

In the next stages of this research, the architecture must be empirically tested to determine whether its theoretical claims are borne out in practice. In the meantime, the contribution of this work is in the field of artificial intelligence, as a theory and implementation of reminding in teaching. We have identified an architecture for establishing instructive situations through a natural question-and-answer dialogue and for capitalizing on those situations with stories. This architecture resembles the way teachers and parents both use stories in their own teaching. In future research, Creanimate can be improved and extended to be a more powerful educational system, and its architecture can be applied to new domains.

Appendix A

The Components of the Indexing Vocabulary

This appendix provides a brief summary of the elements that comprise the Creanimate knowledge representation. It is divided into two sections, one for the objects and one for the relationships.

1. The Creanimate Objects

1.1 Animals

The *animal* object in Creanimate represents a category of animal, as opposed to an individual animal in the world, i.e. the category *cat* not a particular cat. The representation consists of a collection of attributes that all animals in that category share. The values of the slots in an animal object are:

Isa/subtypes. Broader/more specific categories of animal.
Features. Features possessed by all animals in the category.
Actions. Actions performed by all animals in the category.
Behaviors. Behaviors performed by all animals in the category.
Feafuns. The combinations of the features and the actions they support possessed by all animals in the category.
Plans. The combinations of actions and behaviors they achieve possessed by all animals in the category.
Rules. The standard expectations violated by stories that apply to this category of animal.
Stories. Indices of stories in which the animal appears.
Say, say1. Text strings determining how to describe the animal in natural output.
Dmap-parses. Patterns for identifying the animal in student input.

1.2 Feature

A feature is used to represent a category of physical features that animals possess.

Isa/Subtypes. Other more general/specific categories for the same anatomical structure.
Qualities/Quality-Of. Other features with the same physical structure.
Group/Part-Of. Features that this feature is composed of/composes.
Attached-To/Attachments. Other features connected to the feature.
Animals. Animals that possess the feature.

Actions. Actions the feature supports.
Functionality. Abstract actions supported by this feature. Only if feature is abstract.
Required-By. Actions that cannot be performed without this feature.
Suffices-For. Actions that can be performed with just this feature.
Rules. Standard expectations about this feature that are violated by a story.
Stories. Indices of stories which show the use of this feature.
Say, Say1, Say-Its, Say-Their. Text strings corresponding to this feature in different roles in sentences.
Dmap-Parses. Patterns for identifying the feature in student input.

1.3 Action

An action is some basic activity conducted by an animal that can serve more than one survival goal.

Isa/Subtypes. Other more general/specific forms of the action.
Qualities/Quality-Of. Adverbial modifiers of an action.
Part-Of/Parts. Actions that this action composes/is composed of.
Animals. Animals that perform this action.
Behaviors. Behaviors this action achieves.
Features. Features that support this action in some way.
Requires. A list of features that are all essential to perform the action.
Suffices. A list of features any one of which is sufficient to perform the action.
Functionality-For. Abstract features that support this abstract action.
Rules. Standard expectations about the action that are violated by this action.
Stories. Indices of stories that show this action.
Say, Say1, Say-Ing, Say-Ing1, Say-To, Say-To1. Text strings for generating the action in natural language.
Dmap-Parses. Patterns for recognizing the action in student input.

1.4 Behavior

A behavior is a goal-directed activity performed by animals. A behavior can be supported by several different actions.

Achieves. A behavior that is achieved by performing this behavior.
Achieved-By. Other behaviors that achieve this behavior.
Assists. Behaviors that this behavior helps but does not achieve.
Assisted-By. Behaviors that help to acheive this behavior.
Act. A behavior with a more general goal that is an abstract description of all the actions that support the behavior.
Act-Of. A behavior with a more specific goal that shares the act of this behavior.
Part-Of. A behavior composed of this behavior.
Sub-behaviors. The behaviors that compose this behavior.
Isa/Subtypes. More general/specific behaviors. Only used if no other abstraction links apply.
Animals. Animals that perform the behavior.
Actions. Actions that achieve the behavior.
Rules. Default expectations involving the behavior that are violated by stories.
Stories. Indices of stories that show animals performing the behavior.

Say, Say1, Say-Ing, Say-Ing1, Say-To, Say-To1. Text strings corresponding to different ways the behavior can appear in a sentence.

Dmap-parses. Patterns for identifying the behavior in student input.

Object. The object of the verb in the *act* slot. (Used in the patterns in the *dmap-parses* slot.)

1.5 Phys-Obj

The phys-obj object type is used in the *object* slot of behaviors. Phys-obj's are used to represent nouns that may appear in student input but are not animals or features.

Isa/subtypes. More general/specific phys-obj's.

Dmap-parses. Patterns for identifying this phys-obj in student input.

2. The Creanimate Relationships

2.1 Feafun

A feafun represents a feature that is used to perform an action. It is used to connect the pair to an animal that uses the feature in performing the action.

Feature. A feature.

Action. An action the feature supports.

Impact. A subjective description of the relationship. Only used if the feafun appears in an index.

Abstraction. A feafun that contains an abstraction of the feature and action. Only used in feafuns that appear in indices.

2.2 Plan

A plan links an action and a behavior it achieves to an animal that performs them together.

Action. An action.

Behavior. A behavior that the action achieves.

Impact. A subjective impression of the relationship. Only used in indices.

Abstraction. A plan containing an abstraction of the action and of the behavior. Only appears in plans in indices.

2.3 Bplan

A bplan links a behavior with another behavior used in achieving it. Bplans are mainly included for consistency with plans and feafuns because a behavior can only be part of or achieve one other behavior. Therefore, the presence of one behavior in an animal implies the presence of the behavior it is part of or achieves.

Sub. A behavior.

Super. A behavior that the behavior in the *sub* slot either is part of, achieves, or assists.

2.4 All-Rule

An all-rule is used to encode an expectation that a category of animal possesses a feafun, plan or bplan.

Animals. The animals to which the rule applies.

Value. A plan, feafun, or bplan that the animal is expected to possess.

Index. The index of a story that violates the rule.

Expect-viol. The attribute that appears in the story in place of the contents of the value slot.

2.5 Only-Rule

An only-rule is used to encode an expectation that only animals of a particular category possess a feature, action, or behavior.

Animals. The animals to which the rule applies.

Value. A feature, action, or behavior that the animals in the *animals* slot is expected to possess.

Index. The index of a story that violates the rule.

2.6 No-Rule

A no-rule encodes the expectation that no animals in a particular category possess a given feature, action, or behavior.

Animals. The animals to which the rule applies.

Value. A feature, action, or behavior that the animals in the *animals* slot is expected <u>not</u> to possess.

Index. The index of a story that violates the rule.

Appendix B

Sample Creanimate Object Definitions

1. Animals

```
(DEFFRAME animal tree-cricket
     :isa (      [animal jumping-insect] )
     :feafuns (  [feafun     :feature    [feature antennae]
                             :action    [action smell]])
     :plans (    [plan :action    [action smell]
                       :behavior    [behavior detect-
vegetation]] )
     :features ( [feature very-small-size]
                 [feature antennae]
                 [feature mouth-part] )
     :actions ( [action smell] )
     :behaviors ( [behavior detect-vegetation]
                 [behavior protect-eggs] )
     :stories (  [index cricket-smells-tree-cricket] )
     :dmap-parses (LIST (LIST   (list 'tree 'cricket)))
 )

(DEFFRAME animal flying-fox
     :isa (      [animal bat] )
     :feafuns (  [feafun  :feature [feature gripping-claws]
                          :action   [action grip]]
                 [feafun :feature  [feature leathery-wings]
                         :action   [action fan-self]] )
     :plans (    [plan :action    [action fan-self]
                       :behavior   [behavior keep-cool]] )
```

```
     :features ( [feature gripping-claws]
                 [feature leathery-wings]
                 [feature small-size] )
     :actions ( [action fly]
                 [action fan-self]
                 [action grip] )
     :behaviors ( [behavior keep-cool] )
     :stories (  [index bats-cool-off-bat] )
     :say   "fox bats"
     :dmap-parses (LIST (LIST  (list 'flying 'fox)
                               (list 'fox 'bat)))
     :say1  "fox bat"
 )

(DEFFRAME animal amphibian
     :isa (      [animal animal] )
     :subtypes ( [animal frog]
                 [animal toad]
                 [animal salamander] )
     :properties ( [property real-name]
                 [property class] )
     :features ( [feature stomach] )
     :dmap-parses (LIST (LIST (list 'amphibian)))
     :definition  [definition amphibian]
     :complex (  [animal amphibian] )
 )

(DEFFRAME animal frog
     :isa (      [animal amphibian] )
     :subtypes ( [animal lunging-frog]  [animal bufo-frog]
)
     :feafuns (  [feafun    :feature  [feature long-tongue]
                            :action   [action grip]]
                 [feafun    :feature  [feature mouth]
                            :action   [action bite]]
                 [feafun    :feature  [feature eyes]
                            :action   [action look]]
                 [feafun    :feature  [feature ear-drums]
```

```
                  :action   [action listen]]
              [feafun  :feature  [feature webbed-feet]
                     :action [action swim-under-
water]]
              [feafun    :feature  [feature forelegs]
                         :action   [action grip]]
              [feafun    :feature  [feature larynx]
                         :action   [action croak]]
              [feafun    :feature  [feature vocal-sack]
                         :action   [action croak]]
              [feafun    :feature  [feature jaws]
                         :action   [action bite]]
              [feafun :feature [feature strong-hind-
legs]
                     :action   [action leap]]
              [feafun
                   :feature [feature jaws]
                   :action  [action bite-other-
animals]])
     :plans (    [plan :action   [action croak]
                       :behavior   [behavior attract-
mates]]
                 [plan :action   [action grip]
                       :behavior   [behavior grasp-
insects]]
                 [plan :action   [action bite]
                       :behavior   [behavior grasp-
insects]]
                 [plan  :action   [action lunge]
                        :behavior [behavior grasp-insects]]
                 [plan  :action   [action lunge]
                        :behavior   [behavior pounce-prey]]
                 [plan  :action   [action bite]
                        :behavior [behavior grasp-prey]]
                 [plan  :action   [action swim-under-water]
                        :behavior  [behavior flee-
predator]]
                 [plan  :action   [action leap]
```

```
                        :behavior [behavior flee-predator
]]
                  [plan
                      :action   [action look]
                      :behavior [behavior detect-predators]]
                  [plan
                       :action   [action listen]
                       :behavior [behavior detect-
predators]]
                  [plan
                       :action   [action swim-under-water]
                       :behavior [behavior go-to-spawn-
area]]
                  [plan  :action   [action release-sperm]
                         :behavior   [behavior spawn]]
                  [plan  :action   [action lay-eggs]
                         :behavior   [behavior spawn]]
                  [plan  :action   [action grip]
                         :behavior   [behavior spawn]]
                  [plan  :action   [action lunge]
                         :behavior   [behavior grasp-worms]]
                  [plan  :action   [action grip]
                         :behavior   [behavior grasp-worms]]
)
      :features ([feature webbed-feet] [feature forelegs]
                 [feature ear-drums] [feature mouth]
                 [feature long-tongue] [feature larynx]
                 [feature vocal-sack] [feature eyes]
                 [feature jaws] [feature strong-hind-legs]
                 [feature green-color] [feature small-size])
      :actions ( [action bite] [action lunge]
                  [action leap] [action swim-under-water]
                  [action croak] [action listen]
                  [action grip] [action release-sperm]
                  [action look] [action lay-eggs]
                  [action bite-other-animals] )
      :behaviors ([behavior spawn] [behavior grasp-prey]
                  [behavior grasp-insects]
```

```
                [behavior pounce-prey]
                [behavior go-to-spawn-area]
                [behavior detect-predators]
                [behavior flee-predator]
                [behavior attract-mates]
                [behavior grasp-worms] )
     :stories ( [index frog-eats-worm]
                [index frog-and-snake]
                [index grass-frog-mates]
                [index frog-releases-sperm]
                [index frog-eludes-snake-disk-3]
                [index frog-eats-worm-disk-3] )
     :dmap-parses (LIST (LIST (list 'frog)))
)
```

2. Features

```
(DEFFRAME feature tusks
     :isa (     [feature pointy-appendage]
                [feature appendage-on-head]
                [feature teeth]
                [feature long-appendage] )
     :animals ( [animal elephant]
                [animal walrus] )
     :actions ( [action charge]
                [action push-other-animals]
                [action spar] )
     :suffices-for ( [action charge]
                [action push-other-animals]
                [action spar] )
     :stories ( [index elephants-spar] )
     :dmap-parses (LIST (LIST  (list 'tusk)))
)

(DEFFRAME feature nose
     :isa (     [feature appendage-on-head]
                [feature orifice]
                [feature sense-organs] )
```

```
    :subtypes ( [feature beak]
                [feature big-nose] )
    :group (    [feature nostrils] )
    :animals (  [animal land-mammal]  [animal bear]
                [animal squirrel] [animal meerkat] )
    :actions ( [action smell] [action snorkel]
                [action fill-hole] [action dig-hole]
                [action push-dirt] )
    :functionality ( [action dig] [action probe] )
    :properties ( [property one] [property natural-kind] )
    :suffices-for ( [action smell] [action snorkel]
                [action fill-hole] [action dig-hole] )
    :stories (  [index bears-get-food]
                [index squirrel-buries-nuts]
                [index meerkat-digs-hole-meerkat]
                [index tapir-swims-tapir]
                [index ant-bear-roots-insects-ant-bear]
                [index tapir-smells-tapir]
                [index prairie-dog-standby]
                [index squirrel-buries-nuts-disk-3]
                [index bears-get-food]
                [index bears-get-food-disk-3] )
    :dmap-parses (LIST (LIST  (list 'nose)
                              (list 'snout)))
)

(DEFFRAME feature wings
    :isa (      [feature support-in-medium] )
    :subtypes ( [feature feathered-wings]
                [feature large-wings]
                [feature small-wings]
                [feature leathery-wings]
                [feature insect-wings]
                [feature manta-wings]
                [feature medium-size-wings] )
    :qualities ( [feature flat] )
    :actions ( [action fan-self] [action cover-something]
                [action fly] )
```

```
     :functionality ( [action move-in-air] [action shield]
)
     :properties ( [property natural-kind] )
     :required-by ( [action spread-wings]  [action fly] )
     :suffices-for ( [action fan-self]  [action dance]
                    [action cover-something] )
     :stories (  [index bees-fan-hive]
                 [index partridge-nest-and-young]
                 [index butterfly-flies-disk-3] )
     :dmap-parses (LIST (LIST  (list 'wings)
                               (list 'wing)))
     :rules (     [no-rule :animals ([animal land-mammal] )
                           :value    [feature wings]
                           :index [index bats-cool-off-bat]]
                  [no-rule :animals (  [animal fish] )
                           :value      [feature wings]
                           :index [index manta-ray-feeds]]
))
```

3. Actions

```
(DEFFRAME action move
     :isa (      [action action] )
     :subtypes ([action move-fast] [action move-slow]
                [action move-under-water]
                [action move-in-water]
                [action move-self-to-location]
                [action move-in-air]
                [action move-above-ground]
                [action move-on-ground]
                [action move-undetected] )
     :comment    "locomotion only, not move things"
     :dmap-parses (LIST (LIST  (list 'move)
                               (list 'go))))

(DEFFRAME action kick-dirt
     :isa ([action use-force]
```

```
            [action throw] )
      :animals (  [animal kangaroo-rat] )
      :behaviors ( [behavior fight-predators] )
      :features ( [feature legs] )
      :requires ( [feature legs] )
      :stories ([index kangaroo-rat-and-snake-kangaroo-rat])
      :dmap-parses (LIST (LIST  (list 'kick 'dirt)
                                (list 'shoot 'dirt)
                                (list 'throw 'dirt)
                                (list 'kick 'sand)
                                (list 'throw 'sand))))

(DEFFRAME action scrape
      :isa ([action make-contact] [action take-apart])
      :subtypes ([action scrape-at-ground])
      :animals ([animal armadillo] [animal wolf])
      :behaviors ( [behavior build-nest])
      :features ( [feature long-curved-claws]
                  [feature long-sharp-teeth] )
      :suffices ( [feature long-curved-claws]
                  [feature long-sharp-teeth] )
      :functionality-for ( [feature hard] )
      :dmap-parses (LIST (LIST (list 'scrape))))
```

4. Behaviors

```
(DEFFRAME behavior pursue
      :isa ([behavior move] )
      :act-of ( [behavior chase-mate]
                [behavior pursue-prey] )
      :dmap-parses (LIST (LIST  (list 'chase)
                                (list 'run 'after)
                                (list 'pursue)
                                (list 'overtake)))
 )
```

```
(DEFFRAME behavior store-food
     :assists (  [behavior get-food] )
     :actions ( [action deposit-object][action regurgitate]
                [action dig-hole] [action fill-hole] )
     :animals (  [animal bee] [animal squirrel] )
     :object (   [phys-obj food] )
     :stories ([index bees-make-honey]
               [index bees-gather-pollen]
               [index squirrel-buries-nuts]
               [index bees-fight-wasp-disk-3]
               [index bees-gather-pollen-disk-3] )
     :dmap-parses (LIST (LIST  (list 'store ':object)
                               (list 'save ':object)
                               (list 'stock ':object)
                               (list 'stow ':object)
                               (list 'put ':object)
                               (list 'store 'away ':object)
                               (list 'save 'away ':object)
                               (list 'stock 'away ':object)
                               (list 'stow 'away ':object)
                               (list 'put 'away ':object)
                               (list 'put ':object 'away)
                               (list 'put 'it 'away)
                               (list 'store 'it)
                               (list 'save 'it)
                               (list 'stock 'it)))
     :say-ing    "storing food for later"
 )

(DEFFRAME behavior lie-in-wait
     :assisted-by ( [behavior lure-prey] )
     :achieves ( [behavior get-close-to-prey] )
     :actions ( [action remain-motionless]
                [action wade] )
     :animals (  [animal bear]
                 [animal black-heron]
                 [animal pike] )
     :object (   [animal prey] )
```

```
     :stories (  [index salmon-jump-bear]
                 [index black-heron-shades-water] )
     :say        "lie in wait for prey"
     :dmap-parses (LIST (LIST  (list 'wait 'for 'prey)
                               (list 'wait ':object)
                               (list 'sit ':object)
                               (list 'stand ':object)
                               (list 'until ':object)
                               (list 'remain ':object)
                               (list 'patiently ':object)))
     :say-ing    "lying in wait for prey"
 )

(DEFFRAME behavior defend-self
     :subtypes ( [behavior defend-against-predators]
                 [behavior defend-against-enemies] )
     :act ([behavior defend] )
     :say "defend themselves"
     :dmap-parses (LIST (LIST  (list 'defend 'itself)
                               (list 'defend 'themselves)
                               (list 'defend 'self)))
     :say-ing1 "defending itself"
     :say-to1  "to defend itself"
     :say1     "defends itself"
 )
```

Appendix C

Sample Creanimate Index Definitions

The following are examples of indices drawn from the current Creanimate story library.

1. Tiger Captures Deer

In this story, a Bengal tiger chases a deer into a small pond and captures it in its teeth.

```
(DEFFRAME index tiger-captures-deer-tiger
     :animals (  [animal tiger] )
     :features ( [feature mouth] [feature long-legs]
                 [feature long-sharp-teeth] [feature jaws] )
     :behaviors ( [behavior move-food] [behavior grasp-prey]
                 [behavior pursue-prey] )
     :actions ( [action carry-prey] [action run-fast]
                 [action bite] [action bite-other-animals] )
     :feafuns ([feafun  :feature    [feature long-legs]
                        :action   [action run-fast]
                        :impact (   [impact askable] )
                        :abstraction  [feafun  :feature [feature legs]
                                               :action [action move]]]
                 [feafun :feature [feature jaws]
                         :action   [action carry-prey]
                         :abstraction  [feafun
                                            :feature  [feature mouth]
                                            :action [action move-objects]
                                       ]]
                 [feafun :feature    [feature long-sharp-teeth]
                         :action   [action bite-other-animals]
                         :impact (   [impact askable] )
                         :abstraction [feafun :feature [feature mouth]
                                              :action  [action pierce]]
                    ])
     :plans ( [plan :action   [action run-fast]
                    :behavior   [behavior pursue-prey]
```

```
                  :impact (    [impact askable] )
                  :abstraction  [plan :action [action move-on-ground]
                                      :behavior   [behavior hunt]]]
            [plan   :action   [action carry-prey]
                    :behavior   [behavior move-food]
                    :abstraction
                          [plan  :action  [action move-objects]
                                 :behavior [behavior process-food]]]
            [plan      :action   [action bite-other-animals]
                       :behavior   [behavior grasp-prey]
                       :impact ([impact askable :isa [impact askable]
                                            :object [action bite]])
                       :abstraction [plan :action    [action pierce]
                                          :behavior   [behavior hunt]
                                    ]] )
     :story      [story tiger-captures-deer]
     :impact (    [impact great] )
     :rating     8
     :VIDEO (list (list 152802 161000 (disk 'nat-and-brit) "A"))
```

2. Tapir Smells

In this humorous story, a tapir wanders around sniffing the air with its large nostrils flared.

```
(DEFFRAME index tapir-smells-tapir
     :animals (   [animal tapir] )
     :features ( [feature big-nose] [feature nose] )
     :behaviors ( [behavior get-info-world] )
     :actions ( [action smell] )
     :feafuns ([feafun
                :feature    [feature big-nose]
                :action   [action smell]
                :abstraction  [feafun  :feature [feature sense-organs]
                                       :action    [action investigate]]
              ])
     :plans ([plan  :action   [action smell]
                    :behavior   [behavior get-info-world]
                    :abstraction  [plan  :action [action use-senses]
                                         :behavior [behavior get-info]]
              ])
     :story  [story tapir-smells]
     :impact ([impact hilarious] [impact strange-animals] )
     :rating     9
```

```
    :VIDEO (list (list 61101 64100 (disk 'nat-and-brit) "A"))
)
```

3. Cranes Dance

In this story several cranes dance awkwardly on their long legs, hopping, moving their necks, and flapping their wings in an elaborate courtship ritual.

```
(DEFFRAME index cranes-dance-crane
     :animals ([animal whooping-crane] )
     :features ([feature long-legs] [feature legs]
                [feature long-neck] [feature feathered-wings] )
     :behaviors ([behavior attract-mates] )
     :actions ([action spread-wings] [action hop] [action dance] )
     :feafuns ([feafun
                 :feature [feature long-neck]
                 :action   [action dance]
                 :abstraction
                        [feafun :feature  [feature neck]
                                :action [action move-body-parts]]]
               [feafun
                   :feature    [feature feathered-wings]
                   :action   [action spread-wings]
                   :abstraction  [feafun :feature [feature wings]
                                         :action   [action spread]]]
               [feafun
                   feature [feature long-legs]
                     :action    [action hop]
                     :abstraction  [feafun  :feature [feature legs]
                                            :action [action jump]]]
               [feafun :feature [feature feathered-wings]
                       :action [action dance]
                       :abstraction  [feafun
                                        :feature [feature wings]
                                        :action [action be-visible]]]
               [feafun  :feature    [feature long-legs]
                        :action   [action dance]
                        :abstraction
                        [feafun :feature  [feature legs]
                                :action [action be-visible]]])
     :plans ([plan :action   [action spread-wings]
                   :behavior   [behavior attract-mates]
                   :abstraction  [plan :action   [action spread]
                                       :behavior [behavior get-mate]]]
```

```
          [plan :action   [action hop]
                :behavior   [behavior attract-mates]
                :abstraction
                       [plan :action   [action jump]
                             :behavior [behavior attract-mates]]
          [plan :action   [action dance]
                :behavior   [behavior attract-mates]
                :impact (   [impact askable] )
                :abstraction [plan :action [action be-conspicuous]
                                   :behavior [behavior get-mate]]])
     :story [story cranes-dance]
     :impact ([impact frantic-dancing] )
     :rating 5
     :VIDEO (list (list 65201 71700 (disk 'nat-and-brit) "A")))
```

4. Jaja Billabong Shades its Eggs

In this story, a female Billabong bird uses its wings to shield its wings from the searing heat of the sun.

```
(DEFFRAME index jaja-billabong-shades-eggs
     :animals (  [animal jaja-billabong] )
     :features ( [feature feathered-wings] )
     :behaviors ( [behavior protect-eggs] )
     :actions ( [action make-shade] )
     :feafuns ([feafun
                  :feature    [feature feathered-wings]
                  :action   [action make-shade]
                  :abstraction
                      [feafun :feature [feature wings]
                              :action [action shield-from-elements]]
           ] )
     :plans ([plan :action   [action make-shade]
                  :behavior   [behavior protect-eggs]
                  :abstraction
                      [plan :action [action shield-from-elements]
                            :behavior   [behavior care-for-young]]
           ] )
     :story  [story jaja-billabong-protects-eggs-2]
     :expect-fails ([all-rule :animals ( [animal bird] )
                   :value [feafun :feature [feature wings]
                                  :action  [action fly]]
                   :expect-viol [feafun :feature [feature wings]
```

```
                                            :action [action make-shade]]
                        :index [index jaja-billabong-shades-eggs]])
     :impact ([impact funny] )
     :rating 7
     :VIDEO (list (list 302003 303500 (disk 'nat-and-brit) "A"))
)
```

5. Salmon Jump

In this story, salmon swimming upstream to spawn leap many feet into the air to get up and over some rapids.

```
(DEFFRAME index salmon-jump-salmon
     :animals (  [animal salmon] )
     :features ( [feature tail-with-fins]
                 [feature fins]
                 [feature fish-package] )
     :behaviors ( [behavior go-to-spawn-area] )
     :actions ( [action swim-under-water]
                [action jump] )
     :feafuns ([feafun :feature  [feature tail-with-fins]
                    :action   [action jump]
                    :impact (   [impact askable] )
                    :abstraction [feafun
                                   :feature    [feature tail]
                                   :action [action move-above-ground]]]
               [feafun :feature    [feature fish-package]
                    :action   [action swim-under-water]
                    :impact (   [impact askable] )
                    :abstraction
                        [feafun :feature
                                   [feature swim-under-water-package]
                                :action   [action move-in-water]]] )
     :plans ([plan :action   [action swim-under-water]
                   :behavior   [behavior go-to-spawn-area]
                   :impact (   [impact askable :isa ( [impact askable])
                                               :object [action swim]])
                   :abstraction  [plan :action [action move-in-water]
                                       :behavior [behavior mate]]]
             [plan  :action   [action jump]
                    :behavior   [behavior go-to-spawn-area]
                    :impact (   [impact askable] )
                    :abstraction
                              [plan :action [action move-above-ground]
```

```
                                        :behavior [behavior spawn]]])
    :story [story salmon-jump]
    :expect-fails ( [no-rule  :animals ( [animal fish] )
                              :value [action move-in-air]
                              :index [index salmon-jump-salmon]])
    :impact ([impact funny] )
    :rating 9
    :VIDEO (list (list 190706 192229 (disk 'nat-and-brit) "A"))
)
```

Appendix D

User Control Buttons in Creanimate

The User Control Buttons in Creanimate are drawn from a set of sixteen buttons proposed by Jona et al. (1991, see also Schank and Jona 1991). These buttons provide a student with an ability to express his current state in so that an educational system can respond appropriately. Through these buttons, a teaching system can get the sort of information about a student that a human teacher can often read directly from a student's face and actions, and that intelligent tutoring systems devote enormous resources to inferring. In addition, they encourage the student to become more aware of his own learning processes and to act on this awareness. The user control buttons proposed by Jona et al. (1991) fall into three categories: feelings, questions, control. The complete set of sixteen buttons is:

Feelings: Awesome, Boring, No Way, Huh?, Too Hard.

Questions: Why? How do I do that? Now what? What's the point? History?

Control: Change task, Back up, Big picture, More detail, Skip this.

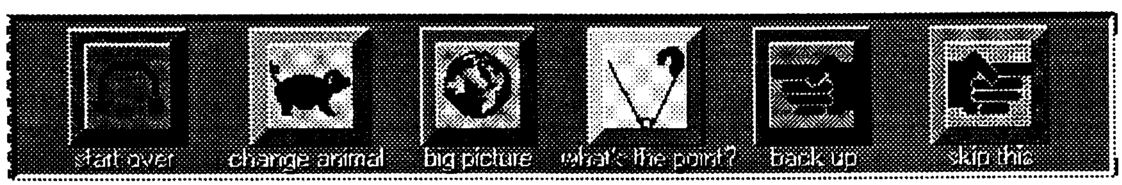

Figure 47. The button palette from the Creanimate user interface showing the User Control Buttons. The buttons are labeled, from left to right, *start over, change animal, big picture, what's the point?, back up,* and *skip this.*

Six user-control buttons were chosen for use in Creanimate. They are shown in figure 47. Their functions are:

Change animal. The *change animal* button gives the student the ability to either start with a new animal or make an additional change to his current animal. The system is designed with the idea that a student should only pursue a topic for as long as it holds his interest. The *change animal* button makes it easy for students to switch to something they find more engaging if their interest in their current animal wanes.

What's the point? The *What's the point?* button is one of two buttons that helps students stay oriented during a dialogue. The *What's the point?* button provides the student with an explanation of why the dialogue manager has said something. Below are two examples of the output from the *What's the point?* button.

```
OK, let's find a reason for your bear to have a beak. Is
there a reason you want your bear to have a beak?
```
Student: Button press-WHAT'S THE POINT?
```
Animals have things for a reason--to help them survive. If
your bear is going to have a beak, it should help it survive
some way.
So, your bear could have a beak to skim the water. Why would
you like your bear to have a beak?
```
Student: Button press-WHAT'S THE POINT?
```
It is your decision why your bear will have a beak. You can
choose now, or see more reasons,and choose later
```

As these examples show, the *What's the point?* button takes a step back and tells a student the reason behind a statement or question from the dialogue manager.

Big Picture. The *Big picture* button helps the student to get his bearings. Because the dialogue is punctuated by videos that can range from 30 seconds to a minute, the student can become so absorbed in a video that he momentarily looses his place. By pressing the *Big picture* button, the student can see a screen that provides information about the dialogue. The *Big picture* screen displays the student's current animal, the question currently under discussion, and questions that have already been discussed. It also displays a list of questions about the animal that the dialogue manager is waiting to discuss. The bottom half of the *Big picture* screen is devoted to a complete transcript of the discussion up to this point. The transcript and the lists of resolved and pending questions are all mouse-sensitive. The student can click on any one of them and jump forward or backward to that point in the dialogue.

Back up. The *back up* button enables the student to go back to any point in the dialogue and do something different. The student can back up one step or as many as he would like. This is empowering for a student because it means he can pursue several alternative paths for a single animal. This helps the student to see the learning interaction not as a series of irrevocable decisions, but as a space of opportunities that can all be explored. The ability to skip to different topics gives the student control over what content he gets exposed to, but the environment is crafted in such a way that the student still gets exposed to the important principles it is teaching. However, the degree of control that a student has allows him to see them in a context that he selects.

Skip this. Like the *back up* button, the *skip this* button allows the student to take control of the flow of the dialogue. In response to the *skip this* button, the dialogue manager will present the student with choices to skip forward within the current dialogue or to skip forward to a pending question. The following transcript reveals the range of choices that a student might see in response to the *skip this* button. In this example, the student is working on a butterfly that can hunt.

```
One part of hunting is catching prey.  How would you like
your butterfly to catch its prey?
```
Student: give me some suggestions
```
Meerkats catch their prey by grasping prey.
I have an interesting video about that. I like this one
because it has scorpions in it.
Would you like to see a video about that?
```

Student: Button Press-SKIP THIS

```
We can do any of the things listed below. Which would you
like to do next?
      "Look at how other animals catch their prey"
      "Talk about how will my butterfly get close to its
prey."
      "Talk about how will my butterfly subdue prey."
      "Talk about how will my butterfly detect prey."
      "Add something else to my butterfly"
      "Start a new animal"
      "Continue where I was"
```

In this example, the student chose to skip a story showing a meerkat catching its prey. The options the dialogue manager presents provide him with an opportunity to skip within the current dialogue and see the ways other animals catch their prey. Or he can skip ahead to other pending questions for his butterfly that fights, such as how it will get close to its prey, subdue its prey, or detect its prey. The final options allow him to change his animal or continue where he was. The *skip this* button allows the student to control the pace and topic of his interaction.

Start over. The *start over* button allows a new student to restart the program after another student has been using it. Pressing the *start over* button erases the program's memory of the current interaction and restores it to the introductory screen.

Appendix E

Transcripts from the Pre-Creanimate Trials

The following transcripts were recorded during experimental sessions conducted in 1987. In these sessions, children believed they were interacting with a computer program but, in fact, they were communicating with William Purves, a biology professor, who was in an adjoining room. Dubbed the "Wizard of Oz" trials, these experiments were conducted to gather data on students' behaviors and to inform the design of the program. The dialogue cycle described in Chapter 6 was modeled on an analysis of these transcripts.

Twenty sessions involving 17 children between the ages of 9 and 12 were conducted. This appendix contains sections of transcripts from three of these sessions.

1. The Experimental Setting

The sessions with children were handled and recorded as follows. The child, accompanied by a parent, is met by Dr. Purves and Maria, his student colleague, and taken (without the parent) to an office with a computer terminal. The child is told a bit about the program, called BILL, on the way to the office. While the college student tells the child about the terminal, Dr. Purves goes to his own office and starts a program running on a DEC VAX to which the child's terminal is also connected. When the child types `BILL` on the terminal, the session begins.

The child sees both his or her own entries and BILL's replies on the screen. A transcript of the session is also maintained by the program. Each line of the dialogue is numbered on the transcript, the speaker (child or BILL) is identified, and each entry is timed, as is the session as a whole. By prior arrangement with the child, the session ends when the child enters `Goodbye*`. Certain parts of the dialogue involve "canned" responses by BILL, but most of the replies are written on the spot by Dr. Purves. At the end of the session, the child is debriefed; he or she is asked to comment on the program, is told that many of BILL's replies were actually provided directly by Dr. Purves, and is allowed to ask any questions about the program and this research project.

During the sessions, Dr. Purves spoke into an audio recorder, keeping track of his thought processes, with particular attention to what inputs trigger the next story to be told. After the session, Purves's remarks are copied into the transcript at the appropriate points; and further comments may be added by Purves or the student.

2. Transcript 1

```
Session of 5 March 1988

ID:             SK
```

SEX: M
AGE: 11
GRADE: 5

~~~~~~

purves:
1
2 Hi!
3
4 I am a very young (and not very smart) computer program.
5 My name is BILL. I would like to talk to you today.
6 When I finish my turn and am waiting for you to say
7 something to me, I will end my message to you
8 with a star. When you finish a thought, you can
9 end your message with a star, too.
...
26 This program works both in the world of nature and in the world of
27 make believe. I can tell you about lots of animals and creatures that
28 exist, or you can think about MAKING UP ANIMALS THAT DON'T EXIST!
29 You might be surprised at how many strange and wonderful animals do exist
30 in nature, like flying frogs, dancing bees, and rodents with no eyes.
31 All these animals live successfully in their particular parts of the world.
...
39 If YOU could make ANY kind of animal, maybe one that is really strange,
40 WHAT WOULD IT BE, or WHAT WOULD YOU WANT IT TO DO? I would really like
41 to hear your ideas!
42 *
student:
43 I WOULD LIKE IT TO FLY LIKE A BIRD*
purves:
44 Okay, that sounds like a good idea!
45 Is it actually a bird, or are we going to
46 build some other animal that flies like a bird? *
student:
47 ITS NOT A BIRD
48 *
purves:
49 Okay. Can you tell me anything more about our
50 new animal, or
51 shall I ask you some questions about it? *
student:
~~~~~~

WELL,IT LOOKS LIKE A DOG AND HAS LARGE EARS
*
purves:
Wow! What an animal!!
It flies like a bird, looks like a dog, and has large ears!
I'd like to hear from you about why it has those large ears.
Could you tell me why our animal has large ears? *
student:
BECAUSE THEY GROW FAST*
purves:
Oh, I see!
What does it DO with those large ears? *
student:
IT MAKES THEM GO UP AND DOWN,
SORT OF LIKE DUMBO
*
purves:
Oh... does that mean our dog-bird-Dumbo uses its ears
to fly and does not have wings? *
student:
YES*
purves:
Okay.
Our animal flies by flapping its ears.
Many birds fly by flapping their wings.
Other birds hold their wings stretched out most of
the time and just soar, without flapping.
Can you think of other ways some other animals fly? *
student:
NO,NOT REALLY
*
purves:
The FLYING FISH doesn't have wings, but it can fly for
a fair distance. The flying fish swims very fast and
then leaps up into the air; it uses broad fins (almost
like wings) to soar for some distance.
The FLYING SQUIRREL has no wings, and no fins. But
when it stretches its front and back legs out to the
sides, they are connected by a web. The flying
squirrel climbs trees, then jumps and stretches out
its legs, and it can soar for some distance.
Their are even some FROGS that can do something like that!

NOTE THAT THIS PROGRAM IS STILL UNDER DEVELOPMENT.
IT WILL HAVE A VIDEO DISK PROGRAM THAT SHOWS PICTURES....
Until I get my video disk, we can only imagine that
I am showing you pictures.
Let's look at another animal that may have big ears.
Look at this photograph of a rabbit that lives in
the icy climate of the Arctic. It has

small, furry ears.
Now look at this photograph of a rabbit that lives near
here. It has larger ears, that aren't so furry.
And now look at this photograph of a rabbit that lives
in the desert. It has VERY large ears, that are
almost hairless!
Can you guess why those different rabbits have ears
that look so different? *
student:
maybe its just nature
*
purves:
That is a reasonable answer!
But I'd like to have you THINK a bit about it...
Those 3 kinds of rabbits live in
3 different climates.
The small, furry ears are in the Arctic,
and the huge, hairless ears are in the desert.
What is a big difference between the Arctic and the desert? *
ones hotter than te other
*
purves:
Yes!!
Now think about something else...
have you ever been REALLY cold, like maybe when
you were in the snow, or in a skating rink, or
just when you were outdoors in the winter? *
student:
yes*
purves:
And what parts of you felt really cold? *
student:
my toes felt numb,and the bottom of my pants
because I fellso many times*
purves:
Ouch!
Lots of people also get very cold EARS unless their
ears are covered up.
When something feels cold like that, it means that
part is LOSING HEAT from the body, so the body gets
colder.
Now let's think about those rabbits...
The ones in the Arctic don't want to lose heat...
they are in a very cold place. Their ears are
small and furry.
The ones in the desert tend to be very hot. The
sun shines on them, and the soil is very hot.
Those rabbits need to LOSE HEAT so they don't
get TOO hot. To do that, they have BIG EARS that
lose heat and cool the rabbit, just as our small

ears lose heat and feel awfully cold in cold weather!

But back to our animal,
that flies with its ears and looks like a dog.
Do you want to tell me anything else about our animal? *
student:
it lives in hot weather and has small feet
purves:
That's very interesting!
Do you want to tell me why the feet are small? *
student:
it has large claws though so it can get food
*
purves:
Okay!
Does it use the claws to DIG for food, or
does it use the claws to CATCH its food? *
to dig for food*
purves:
I see.
Our animal digs for food, and
it can fly, which may help it find its food.
NOTE THAT THIS PROGRAM IS STILL UNDER DEVELOPMENT.
IT WILL HAVE A VIDEO DISK PROGRAM THAT SHOWS PICTURES....
Until I get my video disk, we have to imagine that
I am showing you pictures.
Look at this picture of a dog.
Its eyes look straight ahead.
Look at this picture of a rabbit.
Its eyes point off to the sides of the animal.
Look at this picture of a tiger.
Its eyes look straight ahead.
Look at this picture of a deer.
Its eyes point off to the sides of the animal.
Can you figure out why the eyes point in different
directions on different kinds of animals? *
student:
because of their heads
*
purves:
Do you know the words PREY and PREDATOR? *
student:
yes
*
purves:
Good for you!
Is a tiger a predator, or is it prey? *
student:
its a predator8
*

purves:
Right!
Is a rabbit a predator, or is it prey? *
student:
prey
*
purves:
Very good. You do know those words!
Our animal sounds like a predator to me, if it's
like a dog.
Those pictures showed that the PREDATORS usually
have eyes that look straight ahead of the animal.
The PREY usually have eyes that look off to both
sides.
Can you try to explain that? *
student:
yes because the rabbit needs to be on the look out
for predators that might eat it as a meal
*
purves:
VERY good! You figured it out!
The prey spend their time eating plants, and they
need to be on the look out for predators.
Now why do the predators have eyes that look
straight ahead? *
student:
so they can look for prey
*
purves:
That sounds good to me!
And when they FIND prey, they often must CHASE
the prey. They need to keep their eyes on the
prey, and they aren't worrying about other
predators sneaking up on THEM!

[REMEMBER: you can ask questions about animals any time you want to.
If I have just asked you a question, don't be afraid to ask ME one
right back! Just remember to wait for the beep that says it's
your turn...]

Our animal is going to have long claws, you said, to
help it dig up its food.
Have you decided what the food is that it's digging up? *
student:
im not quite sure do you have any ideas?*
purves:
Well, there are lots of animals that DO dig...
and they often have long claws to help them dig...

animals like woodchucks and badgers and moles...
those animals eat lots of different things, like
worms and insects and so forth.
Do you know the words CARNIVORE and HERBIVORE? *
student:
no
*
purves:
That's okay! A CARNIVORE is an animal that eats other
animals (it eats meat);
an HERBIVORE is an animal that eats plants;
an OMNIVORE is an animal that eats both meat and plants.
We are omnivores.
Is a dog a carnivore or an herbivore? *
student:
a herbivore,I think*
purves:
Think of what dogs like best:
they like bones, and big chunks of meat, and things
like that. Dogs and other predators are CARNIVORES.
Rabbits, cows, deer, garden snails and many
other animals are HERBIVORES.
Back to our digging animal!
Some digging animals eat worms and insects and things
like that--their food is animals, so they are
CARNIVORES.
But some other digging animals, like gophers, eat
plants and are HERBIVORES. These animals may just
eat the roots, but some pull the whole plant down into
their holes and eat the whole thing!

So our animal
looks like a dog,
flies by flapping its ears,
lives in hot weather,
and has small feet with long claws
that it uses to dig for what it eats!
If I had my video disk, I would
try to draw a picture for you...
but I can't.
...

3. Transcript 2

Session of 19 March 1987

ID: AW (SECOND SESSION)
SEX: F =============

AGE: 11
GRADE: 6
TYPING ABILITY: Hunt and Peck...

~~~~~~

purves:
...
26 If YOU could make ANY kind of animal, maybe one that is really
strange,
27 WHAT WOULD IT BE, or WHAT WOULD YOU WANT IT TO DO? I would
really
like
28 to hear your ideas!
29 *
student:
30 I would want it to be a kangaroo.*
purves:
31 That is interesting!
32 Kangaroos do many interesting things.
33 Is there something special about your kangaroo?
34 *
student:
35 Yes it would be able to jump into space.*
purves:
36 That is asking too much of any animal, I think!
37 But I would like to talk about jumping kangaroos...
38 First tell me: how many legs does a kangaroo have? *
student:
39 2
40 *
purves:
41 Okay.
42 It has two strong hind legs
43 and two little front legs that you may think of as arms...
44 It uses only the hind legs to jump and run with.
45 It also has a very big, heavy tail.
46 Can you figure out what it might use that tail for? *
student:
47 To hit things with .*
48 That sounds interesting.
49 In fact, it uses the tail for at least two things:
50 For one thing, it uses the tail to balance itself when
51 it is standing still. It is hard for a 2-legged animal
52 to stand for a long time (at least for SOME 2-legged animals).
53 The kangaroo can lean back slightly on this big tail, and
54 turn itself into a 3-legged stool! (two legs and one tail)
55 ALSO,
56 the kangaroo uses its tail to balance itself when
57 it is running.
58 As the kangaroo runs, its body leans forward. The tail
~~~~~~

sticking out in back keeps it from falling over.
NOTE THAT THIS PROGRAM IS STILL UNDER DEVELOPMENT.
IT WILL HAVE A VIDEO DISK PROGRAM THAT SHOWS PICTURES....
If I had my video disk now I would show you some pictures
of a running kangaroo.

What else about your kangaroo? *
student:
It would be able to sing.*
purves:
What a kangaroo! I wonder what it will sing about...
I am thinking of other animals that sing, such as birds.
What is the reason you would like your kangaroo to sing? *
student:
because kangaroos don`t sing.*
purves:
That means this will be a challenge!
Let's think about animals that DO sing in nature.
Many male birds sing.
They sit in trees or on chimneys and sing and sing.
Do you know WHY they sing? *
student:
to call other birds*
purves:
Yes! They call them for TWO REASONS!
Do you want to figure them out, or
shall I tell you? *
student:
to mate or when their in trouble or when they find food*
purves:
Very good!
The bird has TWO main motives for singing:
they both relate to mating!
The male bird sings to tell OTHER MALE BIRDS:
"This is MY home -- go away!"
The male bird sings to tell FEMALE BIRDS:
"I am a male bird and I am inviting you to visit me!"

But I interrupted your kangaroo story...
you wanted your kangaroo to jump into space
and I said animals couldn't do that
Let's talk for a minute about
animals jumping.
I am thinking of some animals: a kangaroo, a horse, a flea,
a cat, a dog, a deer, and
a little animal called a kangaroo rat.
NOTE THAT THIS PROGRAM IS STILL UNDER DEVELOPMENT.
IT WILL HAVE A VIDEO DISK PROGRAM THAT SHOWS PICTURES....
If I had my video disk, I would show you pictures of
these animals jumping.

They are animals of VERY different sizes! Think of a kangaroo
and a flea.
IN FACT, ALL OF THESE ANIMALS JUMP TO ABOUT THE SAME HEIGHT!
That is, for animals like these, the height to which they can
jump does not depend on the size of the animal!

Why can't a kangaroo or a dog or a flea
jump into space? *
student:
they don`t have enough stranth in their legs to*
purves:
That is exactly right!
We could give them more strength by
giving them bigger muscles.
Think for a minute about a cat with
much bigger muscles, such as
kangaroo muscles...
That would be a stronger cat, and so
you might expect it to be able to jump higher.
But it couldn't... the problem is, the cat
itself would now be bigger!
When an animal jumps, we have to think of
TWO THINGS:
strength and weight!
The muscles give an animal strength, but
they also give it weight. So bigger
animals still jump to about the same height.

Back to the kangaroo...
a kangaroo is a kind of animal found in
Australia.
Like many other Australian mammals, it
is a MARSUPIAL. Do you know that word? *
student:

yes*
purves:
Great!
Then do you know what a MARSUPIUM is? *
student:
no*
purves:
Then I'll tell you about it!
The marsupium is the POUCH that a kangaroo
or other marsupial
carries its babies in.
All marsupials have a marsupium--so that's
where their name comes from.
The marsupium is an interesting thing...
when OTHER mammals have babies, they are

separate from their mother after they
are born. The mother takes care of them
for a while. The babies nurse at their
mother's breasts.
Marsupials are different...
they are born much earlier--they are very
tiny, and they are totally unable to do
anything--even as much as a human baby.
These very tiny babies crawl up along their
mother's fur and climb into the marsupium
(the pouch, or "pocket"). Inside the
marsupium are very tiny nipples. The baby
kangaroo finds a nipple and puts its mouth
over it . . .
and stays there for MONTHS!
So much of its growth is inside the mother's
marsupium instead of inside the mother's womb.
There are many other kinds of marsupials
(besides kangaroos) in Australia.

Shall we keep on talking about
your kangaroo,
or do you want to change the subject? *
keep on talking*
....

4. Transcript 3

Session of 25 February 1987

ID: JB
SEX: M
AGE: 11
GRADE: 6
TYPING ABILITY: Hunt and Peck...

...
If YOU could make ANY kind of animal,
maybe one that is really strange,
WHAT WOULD IT BE, or WHAT WOULD YOU WANT IT TO DO?
I would really like to hear your ideas!
*
student:
IT WOULD BE A SLOGG AND IT WOULD BE LIKE A DOG.*
purves:
Sounds good to me.
Is 'SLOGG' a made-up name, or
do you mean a slug ? Tell me which, please *

student:
IT IS A MADE UP NAME AND I MEAN A SLOGG.*
purves:
Thank you! This ought to be good . . .
Tell me more about our slogg . . .
you said it would be like a dog . . .
how will it be DIFFERENT from an ordinary dog? *
student:
IT CAN FLY.*
purves:
Wow! A flying dog... I mean, a flying slogg!
HOW will it fly? *
student:
IT WILL HAVE WINGS.*
purves:
How many wings will our
slogg
have? *
student:
TWO.*
purves:
Sounds reasonable to me.
Do you know any flying things that have
MORE than two wings? *
student:
YES A DRAGON.*
purves:
I don't know anything about 'DRAGON's . . .
I only know about animals.
Do you know how many wings insects have? *
student:
NO.*
purves:
Most of them have FOUR wings.
Dragonflies have four wings, and so do bugs and beetles.
By the way,
one way we can tell bugs from beetles (they really are
two different kinds of animal!) is by the way their
wings fold up.
In bugs, the wings fold straight down in back, with
the left wings on the left side of the body
and the right wings on the right side of the body.
THAT WAS A MISTAKE.
That was beetles...
In BUGS, the wings cross over each other and look
like an X.

Anyhow, tell me more about our great slogg!
Is there anything else about
it that is different from a dog? *

student:
IT HAS EIGHT LEGS AND TWO ARMS AND WALKS UPRIGHT.*
purves:
Wow! I wonder how it walks upright on eight legs!
Do all 8 of them touch the ground most of the time,
or are some of them hanging up in the air? *
student:
THEY ALL TOUTCH THE GROUND.*
purves:
Hmm... I wish I could draw a picture of this so
we could look at it!
Can you tell me any other kind of animal that
has EIGHT legs? *
student:
A OCTOPUSE BUT THAT IS A MAMAL.*
purves:
YES! an octopus has eight legs! It isn't a mammal,
though. Mammals are things like dogs, cats, bears,
rabbits, mice, people, cows, deer, and so forth.
An octopus is a MOLLUSK. Some other mollusks are
clams, snails, and so forth.
I can think of an animal that has eight walking
legs and two 'arms' with claws on them . . .
I am thinking of a CRAB. A crab is neither
a mammal nor a mollusk. It is an ARTHROPOD.
Other arthropods are insects, lobsters, spiders,
and so forth.

You have said several interesting things.

Let me ask you: do you think dogs would be better
animals if they could fly? *
student:
NO.*
purves:
I agree with you!
A dog with wings could do some things regular dogs
can't, but
they would lose some things, too. It is a lot
of WORK to fly, and the dog would have to eat a
lot more, for example.

So far, our wonderful slogg has two wings, eight
legs, two arms, and stands upright.
Doesn't look a lot like a dog to me any more!

You said our slogg would be like a dog.
In what way is the slogg like a dog? *
student:
A SLOGG HAS THE HEAD OF A DOG.*

purves:
115 Oh, I see!
116 Tell me something about the head of a dog . . .
117 I think you may have a reason for saying that.
118 Is there something special about the dog's head
119 that makes it good for our slogg to have one? *
student:
120 IT HAS GOOD EARS AND A GOOD NOSE TO TRACK DOWN FOOD.*
purves:
121 Yes! It also has good eyes.
122 NOTE THAT THIS PROGRAM IS STILL UNDER DEVELOPMENT.
123 IT WILL HAVE A VIDEO DISK PROGRAM THAT SHOWS PICTURES....
124 If I could make pictures, I would say these things:
125 Look at this picture of a dog --
126 it has eyes on the front of its head.
127 Look at this rabbit --
128 it has eyes on the sides of its head.
129 Look at this picture of a lion --
130 it has eyes on the front of its head.
131 Look at this picture of a deer --
132 it has eyes on the sides of its head.
133 Can you explain this to me? *
student:
134 NO I CAN'T.*
purves:
135 Do you know the words PREY and PREDATOR? *
student:
136 YES.*
purves:
137 Okay, good for you, J!
138 Would you say a lion was usually a PREDATOR, or is it usually PREY? *
student:
139 PREDATOR.*
purves:
140 Good! What about a dog? *student:
141 PREDATOR.*
purves:
142 Very good!
143 What about a rabbit? *
student:
144 PREY.*
purves:
145 You really know your predator and prey stuff!
146 Now think about this...
147 the predators, like lions and dogs and sloggs, have their
148 eyes on the front of their heads (try to picture what they
149 look like);
150 the prey, like rabbits and deer, have their eyes on the
151 sides of their heads (try to picture what they look like).

Now, can you suggest why this might be? That is, how is
it useful for the predator and the prey to have their
eyes where they are? *
student:
NO.*
purves:
Okay. Let's think about this.
Think first about the prey, sitting there chewing grass or
whatever... it is concerned about NOT LETTING PREDATORS SNEAK
UP ON IT. How might it help that prey animal to have
eyes facing to the sides of its head? *
student:
SO THEY CAN SEE THE PREDATOR.*
purves:
Right!
With eyes looking to the sides, it can see almost all the way
around itself, so it is hard to sneak up on a rabbit!
Now, what about the predators, with their eyes looking forward...
can you say anything about that? *
student:
NO.*
purves:
Okay, let me try.
I think that a predator isn't worried about something sneaking
up on it.
It is concerned with chasing and catching things, so it
helps it to have eyes that can point right at what it is
chasing.
In fact other predators besides predator mammals may have
eyes on the fronts of their heads.
Think of an owl, for example.
IF I HAD MY VIDEO DISK,
I'D SHOW YOU A PICTURE OF AN OWL,
SO YOU COULD SEE THAT.

Back to the slogg!
Do you have an interesting reason for it having
eight legs? *
student:
NO*
purves:
Shall we keep on talking about sloggs, or do you
want to build another animal? *
student:
SLOGGS*
purves:
Okay! I LIKE your slogg. Let's see...
it has 2 wings, 8 legs, 2 arms, and it stands upright...
and it has the head of a dog...

191 shall we talk about any of these things, or do you
192 want to tell me OTHER things about the slogg? *
student:
193 Goodbye*
purves:
194 Thank you for talking with me. I REALLY enjoyed it!
195 You have helped the people who are making me smarter.
196 Goodbye, and thanks again!
200 *

References

Ashley, K. D. and V. Aleven. 1992a. "Automated generation of examples for a tutorial in case-based argumentation." In Proceedings of the Second International Conference on Intelligent Tutoring Systems in Montreal,

Ashley, K. D. and V.Aleven. 1992b. "Generating dialectical examples automatically." In AAAI-92: Proceedings of the Tenth National Conference on Artificial Intelligence in San Jose, CA. 654-660. Menlo Park: AAAI Press.

Bareiss, R. and R. Osgood. 1993. "Applying AI models to the design of exploratory hypermedia systems." Paper presented at the annual meeting of the American Association for Educational Research, Atlanta, Georgia, 12-16 April 1993..

Bassett, G. W. 1970. Innovation in primary education: A study of recent developments in primary education in England and the U.S.A. London: John Wiley & Sons.

Bell, B. L. and R. Bareiss. 1993. "Sickle cell counselor: Using goal-based scenarios to motivate the exploration of knowledge in a museum context." In Proceeedings of the World Conference on AI in Education in Edinburgh. Charlottesville, VA: Association for the Advancement of Computers in Education.

Brachman, R. J. and H.J. Levesque, eds. 1985. Readings in knowledge representation. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Braitenberg, V. 1984. Vehicles: Experiments in synthetic psychology. Cambridge, Mass.: MIT Press.

Brown, A. L. and A. S. Palincsar. 1989. Guided, cooperative learning and individual knowledge acquisition. In Knowing, learning, and instruction: Essays in honor of Robert Glaser, ed. L. B. Resnick. 393-451. Hillsdale, NJ: Lawrence Erlbaum Associates.

Brown, J. S., A. Collins, and P. Duguid. 1989. SItuated cognition and the culture of learning. Educational Researcher 18 (1): 32-42.

Bryant, S. C. 1905. How to tell stories to children. Cambridge, Mass.: Houghton Mifflin.

Burke, R. and A. Kass. 1992. "Integrating case presentation with simulation-based learning-by-doing." In Fourteenth Annual Conference of the Cognitive Science Society in Bloomington, IN, 629-634. Lawrence Erlbaum Assoc.

Burton, R. R. and Brown, J. S. 1982. Investigation of computer coaching for informal learning activities. In Intelligent tutoring systems, ed. D. Sleeman and J. S. Brown, 79-97. London: Academic Press.

Carbonell, J. R. 1970a. AI in CAI: An artificial intelligence approach to computer-assisted instruction. IEEE Transactions on Man-Machine Systems 11 (4): 190-292.

Carbonell, J. R. 1970b. Mixed-initiative man-computer instructional dialogues. Doctoral dissertation. Massachusetts Institute of Technology.

Charniak, E., C. K. Riesbeck, D. V. McDermott, and J. R. Meehan. 1987. Artificial intelligence programming. Second ed., Hillsdale, NJ: Lawrence Earlbaum Associates.

Chi, M. T. H., M. Bassok, M. W. Lewis, P. Reimann, and R. Glaser. 1989. Self-explanations: How students study and use examples in learning to solve problems. Cognitive Science 13: 145-182.

Clancy, W. J. 1987. Knowledge-based tutoring: The GUIDON program. Cambridge, MA: MIT Press.

Clancy, W.J., E.H. Shortliffe, and B.G. Buchanan. 1984. Intelligent computer-aided instruction for medical diagnosis. In Medical artificial intelligence: The first decade, ed. W.J. Clancey and E.H. Shortliffe. Reading, Massachusetts: Addison-Wesley.

Cleave, J. B., D. C. Edelson, and R. Beckwith. 1993. "A matter of style: An analysis of student interaction with a computer-based learning environment." Paper presented at the annual meeting of the American Educational Research Association, Atlanta, Georgia, 12-16 April 1993.

Coderre, B. 1988. Modeling behavior in petworld. In Artificial life, ed. C. Langton. 407-419. Reading, Mass.: Addison-Wesley.

Cognition and Technology Group at Vanderbilt. 1990. Anchored instruction and its relationship to situated cognition. Educational Researcher 19 (6): 2-10.

Cole, B. 1993. "The technology framework." IEEE Spectrum, March 1993 1993, 32-39.

Coles, R. 1989. The call of stories: Teaching and the moral imagination. Boston: Houghton Mifflin.

Collins, A., J. S. Brown, and D. E. Newman. 1989. Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In Knowing, learning, and instruction: Essays in honor of Robert Glaser, ed. L.B. Resnick. 453-494. Hillsdale, NJ: Lawrence Erlbaum Associates.

Collins, A. and A. L. Stevens. 1982. Goals and Strategies of Inquiry Teachers. In Advances in instructional psychology II, ed. R. Glaser. Hillsdale, NJ: Lawrence Earlbaum Associates.

Copeland, M. T. 1954. The genesis of the case method in business instruction. In The case method at the harvard business school: Papers by present and past members of the faculty and staff, ed. Malcolm P. McNair. New York: McGraw-Hill Book Company, Inc.

De Wit, D. 1979. Children's faces looking up : Program building for the storyteller. Chicago: American Library Association.

Deci, E.L. 1975. Intrinsic motivation. New York: Plenum.

Dillon, J.T. 1985. Using questions to foil discussion. Teaching and teacher education 1 (2 1985): 109-121.

Edelson, D.C. 1992. The Creanimate Knowledge Representation Manual. Institute for the Learning Sciences, Northwestern University. Unpublished manual.

Feigenbaum, E. A. 1977. "The art of artificial intelligence: Themes and case studies of knowledge engineering." In IJCAI 5: Proceedings of the Eighth International Joint Conference on Artificial Intelligence.1014-1029.

Feynman, R. P., R. B. Leighton, and M. Sands. 1963. The Feynman lectures on physics. Reading, Mass.: Addison-Wesley.

Fikes, R. and N. Nillson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2 (1971): 189-208.

Fitz-Gerald, C. and D. Gunter. 1971. Creative storytelling, for library and teacher aides. Dallas: Leslie Press.

Fitzgerald, W. A. 1992. Direct memory access parsing in the creanimate biology tutor. Institute for the Learning Sciences, Northwestern University, Unpublished Manuscript.

Forman, G. and P. Pufall, eds. 1988. Constructivism in the computer age. Hillsdale, NJ: Lawrence Earlbaum Associates.

Foster, J. 1972. Discovery learning in the primary school. London: Routledge & Kegan Paul.

Gonella, J.S., M.J. Goran, J.W. Williamson, and M.J. Cotsonas. 1970. Evaluation of patient care. Journal of American Medical Association 214 (11): 2040-2043.

Guha, R. V. and Douglas B. Lenat. 1990. CYC: A midterm report. AI Magazine 11 (3): 33-59.

Halloun, I. A. and D. Hestenes. 1985. The initial knowledge state of college physics students. American Journal of Physics 53 (11): 1043-1055.

Hammond, K. J. 1986. Case-based planning: An integrated theory of planning, learning and memory. Yale University.

Harter, S. 1981. A new self-report scale of intrinsic versus extrinsic orientation in the classroom: Motivational and informational components. Developmental Psychology 17 (1981): 300-312.

Jona, M., B. Bell, and L. Birnbaum. 1991. "Button theory: A taxonomy of student-teacher communication for interface design in computer-based learning environments." In Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society in Chicago, IL, 765-769.

Jones, A. and J. Buttrey. 1970. Children and stories. Oxford: Blackwell.

Karakotsios, K. 1992. SimLife. Maxis. Computer Program.

Kass, A., R. Burke, E. Blevis, and M. Williamson. 1992. The GuSS project: Integrating instruction and practice through guided social simulation. Institute for the Learning Sciences, Northwestern University, Technical Report 34.

Keller, J. M. 1983. Motivational design of instruction. In Instructional-design theories and models: an overview of their current status, ed. Charles M. Reigeluth. 383-434. Hillsdale, NJ: Lawrence Earlbaum Associates.

Klein, G.A. and Calderwood, R. 1988. "How do people use analogues to make decisions?" In Proceedings: Case-Based Reasoning Workshop (DARPA) in San Mateo, CA, 209-218. Morgan Kauffmann Publishers.

Kolodner, J. L. 1991. Improving human decision making through case-based decision aiding. AI Magazine 12 (2): 52-67.

Kolodner, J. L., R. L. Simpson, and K. Sycara-Cyranski. 1985. "A process model of case-based reasoning in problem-solving." In Ninth International Joint Conference on Artificial Intelligence.

Leinhardt, G. 1987. "Situated knowledge and expertise in teaching." In Teacher's Professional Learning in Lewes, Essex, England, edited by J. Calderhead, Falmer Press.

Lenat, D. B. and R. V. Guha. 1990. Building large knowledge-based systems. Reading, Mass.: Addison-Wesley.

Lepper, M. R. and D. Green, eds. 1978. The hidden costs of reward: new perspectives on the psychology of human motivation. Hillsdale, NJ: Lawrence Erlbaum Associates.

Lepper, M. R., D. Greene, and R. E. Nisbett. 1973. Undermining children's intrinsic interest with extrinsic rewards: A test of the "overjustification" hypothesis. Journal of Personality and Social Psychology 28: 129-137.

Lepper, M. R. and M. Hodell. 1989. Intrinsic motivation in the classroom. In Research on motivation in education, 73-105. London: Academic Press, Inc.

Levine, H.G. and P.M. Foreman. 1973. A Study of Retention of Knowledge of Neurosciences Information. Journal of Medical Education 48 (1973): 867-869.

Lindsay, R. K., B. G. Buchanan, E.A. Feigenbaum, and J. Lederberg. 1980. Applications of artificial intelligence for organic chemistry: The DENDRAL project. New York: McGraw-Hill.

Malone, T. 1981. Toward a theory of intrinsically motivating instruction. Cognitive Science 4 : 333-369.

Malone, T. W. 1980. What makes things fun to learn: A study of intrinsically motivating computer games. Doctoral dissertation, Stanford University.

Malone, T. W. and M. R. Lepper. 1987. Making learning fun: A taxonomy of intrinsic motivations for learning. In Aptitude, learning, and instruction: Vol. III. Conative and affective process analyses, ed. R. E. Snow and M. J. Farr. Hillsdale, NJ: Lawrence Earlbaum Associates.

Martin, C. 1989. Case-based parsing. In Inside case-based reasoning, ed. C. K. Riesbeck and R. C. Schank. Hillsdale, NJ: Lawrence Erlbaum Associates.

Martin, C.E. 1990. Direct memory access parsing. Ph.D. dissertation, Yale University, New Haven, CT.

McCloskey, M., A. Caramazza, and B. Green. 1980. Curvilinear motion in the absence of external forces: Naive beliefs about the motion of objects. Science 210: 1139-1141.

McConnell, J. V. 1977. Understanding human behavior. Second ed., New York: Holt, Rhinehart & Winston.

McConnell, J. V. 1978. Confessions of a textbook writer. American Psychologist 33 (1): 159-169.

National Geographic Society. 1989. Mammals: A multimedia encyclopedia. Washington, DC: National Geographic Society. CD-ROM.

Ohmaye, E.. 1992. Simulation-based language learning: An architecture and a multimedia authoring tool. Doctoral Dissertation, Northwestern University.

Optical Data Corporation. 1989. Encyclopedia of animals/The living textbook. Interactive Videodisc.

Paley, V. G. 1990. The boy who would be a helicopter: The uses of storytelling in the classroom. Cambridge, Mass.: Harvard University Press.

Papert, S. 1980. Mindstorms: children, computers, and powerful ideas. New York: Basic Books, Inc.

Papert, S. 1986. Constructionism: a new opportunity for elementary science education. MIT Media Laboratory, Proposal to the National Science Foundation. Quoted in Mitchell Resnick. "Beyond the Centralized Mindset." In The International Conference on the Learning Sciences: Proceedings of the 1991 Conference in Evanston, Ill., edited by L. Birnbaum, 389-396. Charlottesville, VA: Association for the Advancement of Computing in Education. 1991.

Patterson, S. L. 1990. Designasaurus II. San Francisco, CA: Britannica Software. Computer Program.

Piaget, J. 1954. The construction of reality in the child. New York: Basic Books.

Quillian, M. R. 1966. Semantic memory. Doctoral dissertation, Carnegie Institute of Technology.

Redlich, J. 1914. The common law and the case method in American university law schools. The Carnegie Foundation for the Advancement of Teaching, Bulletin Number 8.

Reed, A. Z. 1921. Training for the public profession of the law. Carnegie Foundation for the Advancement of Teaching, Bulletin Number 8.

Resnick, M. 1988. LEGO, Logo, and life. In Artificial Life, ed. C. Langton. 407-419. Reading, Mass.: Addison-Wesley.

Resnick, M. 1991a. "Animal simulations with *Logo: Massive parallelism for the masses." In From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior in edited by J. A. Meyer and S. W. Wilson, 534-539. MIT Press.

Resnick, M. 1991b. "Beyond the centralized mindset." In The International Conference on the Learning Sciences: Proceedings of the 1991 Conference in Evanston, Ill., edited by L. Birnbaum, 389-396. Charlottesville, VA: Association for the Advancement of Computing in Education.

Riesbeck, C. K. and R. C. Schank. 1989. Inside case-based reasoning. Hillsdale, NJ: Lawrence Erlbaum Associates.

Ross, B.H. 1989. "Some Psychological results on case-based reasoning." In Case-Based Reasoning: Proceedings of a Workshop on Case-Based Reasoning in San Mateo, CA, 144-147. Morgan Kaufmann Publishers.

Sacerdoti, E. 1977. A structure for plans and behavior. American Elsevier.

Schank, R., W. Ferguson, M. Brand, R. Burke, E. Domeshek, D. Edelson, M. Freed, M. Jona, B. Krulwich, E. Ohmaye, R. Osgood, and L. Pryor. 1990a. "Beyond process: a universal content theory for indexing." In AAAI Spring Symposium Series, Symposium on case-based reasoning in Stanford, CA,

Schank, R., R. Osgood, M. Brand, R. Burke, E. Domeshek, D. Edelson, W. Ferguson, M. Freed, M. Jona, B. Krulwich, E. Ohmaye, and L. Pryor. 1990b. A content theory of memory indexing. Institute for the Learning Sciences, Northwestern University, March 1990. Technical Report 2.

Schank, R. .C. 1982. Dynamic memory. Cambridge: Cambridge University Press.

Schank, R. C. 1986. Explanation patterns: understanding mechanically and creatively. Hillsdale, NJ: Lawrence Erlbaum Associates.

Schank, R. C. 1990. Tell me a story: a new look at real and artificial intelligence. New York: Charles Scribner's Sons.

Schank, R. C. 1991. Case-based teaching: Four experiences in educational software design. Institute for the Learning Sciences Technical Report, Northwestern University, Technical Report 7.

Schank, R. C. and R. P. Abelson. 1977. Scripts, plans, goals, and understanding. Hillsdale, NJ: Lawrence Erlbaum Associates.

Schank, R. C., R. Bareiss, A. Fano, R. Osgood, and W. Ferguson. 1992. Agents in the story archive. Institute for the Learning Sciences, Northwestern University, May 1992. Technical Report 27.

Schank, R. C. and M. Y. Jona. 1991. Empowering the student: New perspectives on the design of teaching systems. The Journal of the Learning Sciences 1 (1): 1-34.

Schoenfeld, A. H. 1992. Learning to think mathematically: Problem solving, metacognition, and sense-making in mathematics. In Handbook of research on mathematics teaching and learning, ed. D. A. Grouws. New York: Macmillan.

Shortliffe, E. H. 1976. Computer-based medical consultation: MYCIN. New York: American Elsevier.

Shulman, L. S. and E. R. Keisler, eds. 1966. Learning by discovery. Chicago: Rand, McNally.

Sleeman, D. H. and J. S. Brown, eds. 1982. Intelligent tutoring systems. London: Academic Press.

Steele, Guy L., Jr. 1990. Common Lisp: The Language. Second ed., Bedford, Mass.: Digital Press.

Stevens, A.L. and A. Collins. 1977. "The goal structure of a socratic tutor." In Proceedings of the National ACM Conference in Seattle Washington, 256-263. New York: Association for Computing Machinery.

Symbolics Inc. 1991. Common lisp interface manager (CLIM): Release 1.0. Symbolics, Inc., March 1991. Reference Manual.

Thorndyke, P. W. 1975. Cognitive structures in human story: Comprehension and memory. Santa Monica, CA: Rand Corp.

Travers, M. 1988. Animal construction kits. In Artificial Life, ed. C. Langton. 421-442. Reading, Mass.: Addison-Wesley.

Wenger, E. 1987. Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge. Los Altos, CA: Morgan Kaufmann.

Whitehead, A. H. 1929. The aims of education. Cambridge: Cambridge University Press.

Williams, S. M. 1991. Putting case-based instruction into context: Examples from legal, business, and medical education. Learning Technology Center, Vanderbilt University, November 1991. Technical Report

Winne, P. H. 1979. Experiments relating teachers' use of higer cognitive questions to student achievement. Review of Educational Research 49 (1): 13-50.