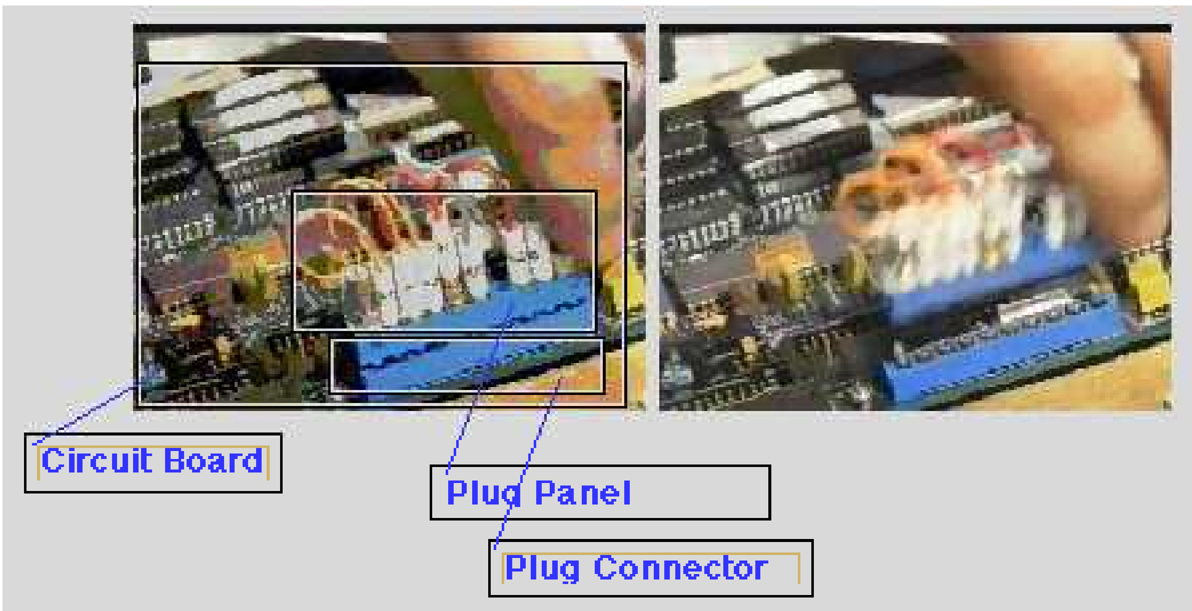
Conference of the American Association for Artificial Intelligence, Seattle, 31 July - 4 August 1994

A User Interface for Knowledge Acquisition From Video

Henry Lieberman

Media Laboratory
Massachusetts Institute of Technology
Cambridge, Mass. USA
lieber@media.mit.edu



Abstract

In conventional knowledge acquisition, a *domain expert* interacts with a *knowledge engineer*, who interviews the expert, and codes knowledge about the domain objects and procedures in a rule-based language, or other textual representation language. This indirect methodology can be tedious and error-prone, since the domain expert's verbal descriptions can be inaccurate or incomplete, and the knowledge engineer may not correctly interpret the expert's intent.

We describe a user interface that allows a domain expert who is not a programmer to construct representations of objects and procedures directly from a video of a human performing an example procedure. The domain expert need not be fluent in the underlying representation language, since all interaction is through direct manipulation. Starting from digitized video, the user selects significant frames that illustrate before- and after-states of important operations. Then the user graphically annotates the contents of each selected frame, selecting portions of the image to represent each part, labeling the parts, and indicating part/whole relationships. Finally, programming by demonstration techniques describe the actions that represent the transition between frames. The result is object descriptions for each object in the domain, generalized procedural descriptions, and visual and natural language documentation of the procedure. We illustrate the system in the domain of documentation of operational and maintenance procedures for electrical devices.

Keywords: Graphical annotation, knowledge acquisition, knowledge representation, machine learning, multimedia, natural language generation, programming by example/by demonstration, user interfaces.

Video as a tool for procedural knowledge representation

Conventional expert systems are often hampered by the knowledge acquisition bottleneck: the difficulty of communicating knowledge that a human expert has into computer-readable descriptions. Domain experts may already know how to perform a procedure when presented with a real-world example of the device, but they may not be capable of producing an accurate, complete, well-written description of the procedure, either in natural language, or in the form of frame descriptions and if-then rules. The role of a knowledge engineer is to interview the domain expert and perform the translation into a language that the domain expert typically does not know. However, this is difficult and

Four frames from a disassembly procedure

unreliable, as the expert can have difficulty verbally expressing the procedure and the knowledge engineer may misinterpret the expert's description. Technical documentation writers and illustrators watch examples and listen to vague descriptions and produce precise, well-written and well-illustrated procedural descriptions that can be understood by a novice.

In both cases, video often plays an important role as a tool to collect the raw data used in constructing the procedural description. The experts are recorded on video, and the resulting video is examined, often in excruciating detail, by those producing the description. It can be helpful to work from video rather than rely on memory or a written description, since it is often awkward to write down or sketch each step while in the midst of performing the procedure. Psychologists, industrial designers, and user interface consultants all make use of videotaped procedures.

With the advent of digital video, it is now possible to provide on-line tools that let the user browse video segments and construct descriptions. Tools can assist in setting up the correspondence between video frames and steps of the procedure, selecting possible descriptions automatically, and providing better feedback that the description actually matches the procedure. The result should be reducing the cost and time required for knowledge acquisition and improving the accuracy of the resulting description.

Machine learning from video demonstrations

Machine learning (Carbonell 90) has extensively studied the problem of inferring general descriptions from specific examples. Most often, these examples are expressed with textual assertions in a formal language or natural language. However, because of the conceptual distance for many users between examples performed in the real world on physical objects and textual descriptions of the examples, much of the work on machine learning has been difficult to apply in the context of interactive interfaces. This has also been a criticism of the *situated action* school (Barwise and Perry 83).

A powerful technique for going from examples to procedural descriptions is *programming by demonstration* or *programming by example*. A compendium of major work on this topic is (Cypher, ed. 93). In this approach, a user demonstrates a procedure in an interactive graphical interface, using concrete visual examples, while a learning mechanism records user actions so that they can be generalized to work on similar examples in the future.

If we imagine that we had already programmed a simulation of a device and the vocabulary of operations for using it, programming by demonstration becomes an excellent tool for producing procedural descriptions. The expert can go through the steps of the procedure on the virtual device, and the system records the descriptions automatically. Examples of this approach appear in (Lieberman 92, 93b).

However, producing the domain-specific representations of the objects and actions in the simulation is expensive and time-consuming, worthwhile if many different procedures are to be performed using the same set of objects and actions, but costly for a single use.

Here, we explore the approach of working directly from a video of the procedure, performed in the real world rather than on virtual objects. But the video itself has no knowledge of where the objects are in the scene or how the actions are to be described. So we require an *annotation* phase, where the author of the description explicitly indicates to the system where the objects are in the video and how the actions are to be described. Graphical annotation (Lieberman 93b) is used to describe objects, and programming by demonstration techniques describe the actions. The system is implemented as an extension to the programming by demonstration system Mondrian (Lieberman 93a).

In the future, computer vision algorithms may be able to automatically detect objects and motion in the video, which would reduce the tedium of the annotation steps. At present, the system uses no visual recognition, but recognition algorithms could be added to the system incrementally, and the user could always retain the option of reverting to explicit description if the vision system guesses incorrectly.

Generality of the approach

The task of generalizing from specific examples is always underconstrained; the learner must assume some bias if nontrivial generalizations are to be learned. For our domain of simple equipment maintenance procedures we have developed an ontology that is roughly characterized as follows.

The procedure learned by the system are applicable to physical *devices*, which are composed of hierarchies of *parts*. Parts may be have specific types, such as switches or fasteners. Each part is represented visually by an image which is a rectangular region of a video frame. There are a set of generic actions which are applicable to devices and parts: assembly and disassembly of parts, opening and closing fasteners, turning on and off switches. In addition, the devices are embedded in a graphical editor with generic cut, copy, and move actions, and relations such as above, below, left and right. The job of the knowledge acquisition phase is to identify device and part hierachies, and specify a sequence of actions upon a device.

The set of actions, objects and relations is extensible. There is an underlying object-oriented protocol for adding new object types, actions, relations and generalizations, which can be used to adapt the video annotation interface to a new domain. The video annotation interface is currently customized for the domain described above, but we have also experimented in other domains. A further development would be to allow extending the domain through graphical interaction in the interface itself.

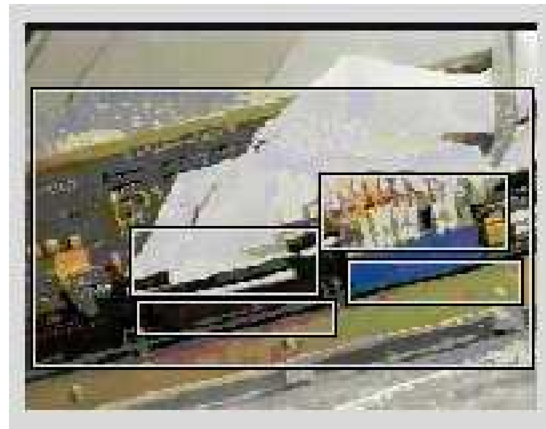
An example: Describing a disassembly procedure

The illustration above shows four frames from a video of a circuit board disassembly procedure. We use QuickTime digitized video (Apple 93), and the standard QuickTime video browsing tools to let the user peruse and select frames from the video. For each step that the user wishes to describe, he or she must select salient frames that represent the states before and after the relevant action. The system provides a tool that "grabs" the frame from the video.

Identifying objects in the video frame

We'd like to give the user the illusion of interacting with a simulation of the device that is the subject of the procedure. If we were to program such a simulation from scratch, the programmer would produce renderings of each object, and operating on these objects would produce a corresponding change in the renderings to reflect the consequences of the action. But that assumes the underlying model of the device and its structure already exists. In our video annotation world, the purpose of the interaction is exactly to teach the system about the object model of the device. But the video contains only pixels, not a 3D object model. So we "fake" it.

We ask the user to select subsets of the video image to serve as visual representations of the objects. The image objects are embedded in a two-and-a-half dimensional graphical editor. Operations of the 2.5D graphical editor approximate the effect of true 3D operations.



Objects segmented from a video frame

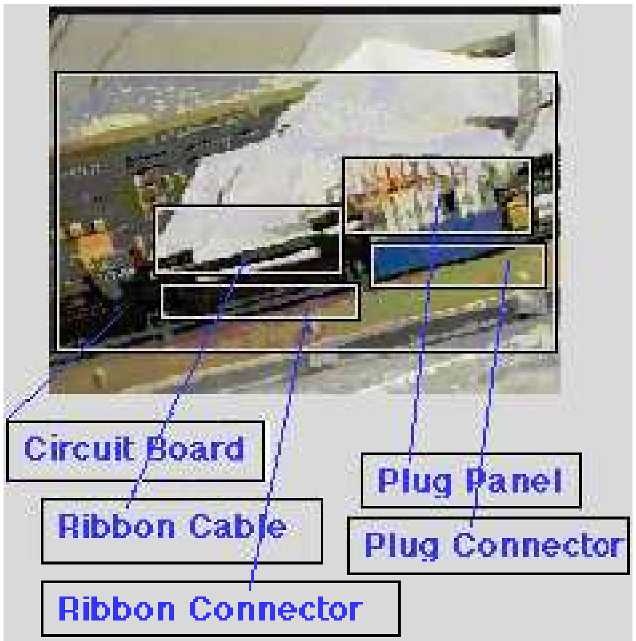
Of course, this "doesn't work" -- the image boundaries capture unwanted pixels and chop off desired ones, moving an image doesn't properly show occlusion and perspective, etc. But it is only meant to be a visual suggestion of the effect of the operation rather than an accurate portrayal. Like icons, the approximate visual representation can be remarkably effective at times.

In our technical documentation domain, there are certain classes of operations that are quite amenable to this treatment. Many of the procedures are assembly and disassembly procedures. Adjacency of the visual representation of two parts indicates that they are assembled, and the act of separating them by moving one of the parts constitutes disassembly. Many of the operational procedures involve state changes: turning a switch on or off, a dial that can be in any one of a number of states, indicator lights. Such operations are modeled by choosing an image representing each state, and replacing one with another to change state.

Graphical annotation of frames

The next step is to label the images with a description of the objects and their structure. Our approach is to use *graphical annotation* of the image. In hardcopy technical manuals, a common sight is a labeled picture or drawing of an object. The name of each part of the object appears as text, positioned as close as possible to the part that it describes, and with a line or arrow connecting the label to its referent.

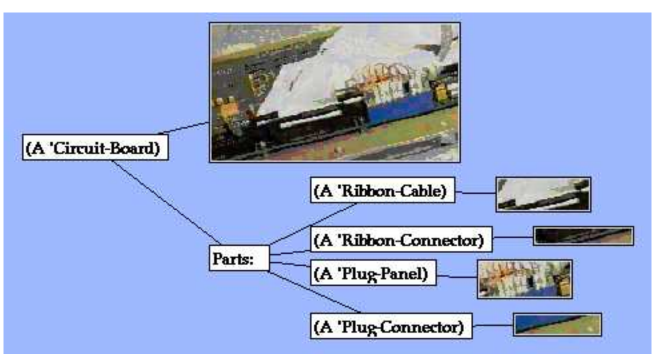
Mondrian allows the user to annotate each video frame image. Text and lines are positioned using the ordinary capabilities of the graphical editor. The user can group a set of text labels as a way of indicating that the corresponding parts they label should be considered as parts of a larger structure. Groups of groups can create part/whole hierarchies of arbitrary depth.



Parts annotated with text labels

A simple visual parser relates the text and lines that do the annotation to the images of the parts they annotate. The visual parser maps grouping and containment relations of the graphical objects to part/whole relations on the semantic objects they represent. Our simple parser has a single fixed "visual grammar", but see (Lakin 86) and (Wittenburg and Weitzman 93) for more sophisticated parsing techniques. The annotation aspect of Mondrian is discussed further in (Lieberman 93b).

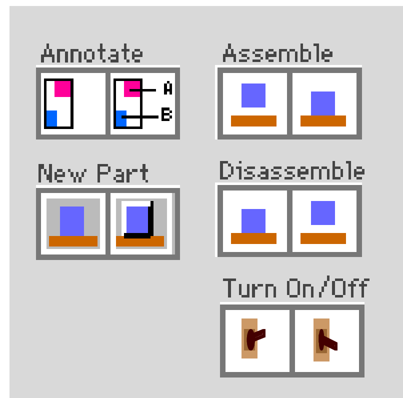
We use a part/whole graph to give the user visual feedback about the results of the visual parsing process. This consists of a graph whose leaves are the images that represent the parts, and whose non-terminal nodes are the names of the parts and wholes.



Part/Whole graph

The effect of the annotations is to attach a description to each part that affects how actions involving it are generalized by the programming by demonstration system. By attaching the label "Ribbon Cable" to an image, the description "the Ribbon Cable of the Circuit Board" is attached to that image, by virtue of the part/whole hierarchy. Any operation applying to that image will be generalized as applying to whatever part is identified as the Ribbon Cable in whatever part corresponds to the Circuit Board in subsequent examples.

The text and lines that comprise the annotations are formed into a group as the result of the visual parsing process so they may be deleted or moved as a unit. This will be useful in making it convenient to apply the same set of annotations to a different frame.



Some operations for image description (left) and repair procedure domain (right)

Descriptions of actions

The next task is to describe the actions that make up the steps of the procedure. Again, since we can't at present detect these transformations using computer vision, we rely on the user to describe the procedural transformation.

Each step consists of an initial state and a final state, represented by selected frames of the video. After annotating the objects in the frame representing the initial state, the user selects a frame to represent the outcome of the operation. To construct a description of the procedure step, the user must use the operations of the graphical editor to transform the initial state to (an approximation of) the final state.

The editor provides both generic operations (Cut, Copy, Paste, Move, ...) and operations specific to the domain (for the repair domain: Assemble, Disassemble, Turn On/Off switches and lights, Open/Close fasteners, etc.). Each operation is represented on-screen by a *domino*, an icon representing a visual example of the operation. The domino has two parts, showing schematized before-and-after states of an example of the operation.

The illustration shows three frames. First, an initial frame, with four annotated objects. In the middle frame, we have applied the Disassemble operation to the object at the upper left, moving it away from its connector. This models the goal frame at the right, where the hands of the technician can be seen disconnecting the cable. The system records a Lisp expression for this step that could be described this way: "Disassemble the Ribbon-Cable of the Circuit-Board from the Ribbon-Connector of the Circuit-Board".

The system tracks the dependencies between operations, so that if the same cable were subsequently connected to another connector, that fact would be noted. If the circuit board appears as the argument to the procedure being defined, the Disassemble operation would be applied to the Ribbon-Cable of whatever Circuit-Board is supplied in the next example.

Once a step of the procedure has been described, we can prepare to describe the next step. To ensure the continuity of the procedure, we need to *transfer* the description of the first initial frame to the next. A priori, the system cannot know which portions of the new image correspond to objects in the new frame, since the objects may have moved, camera angle may have changed, etc.

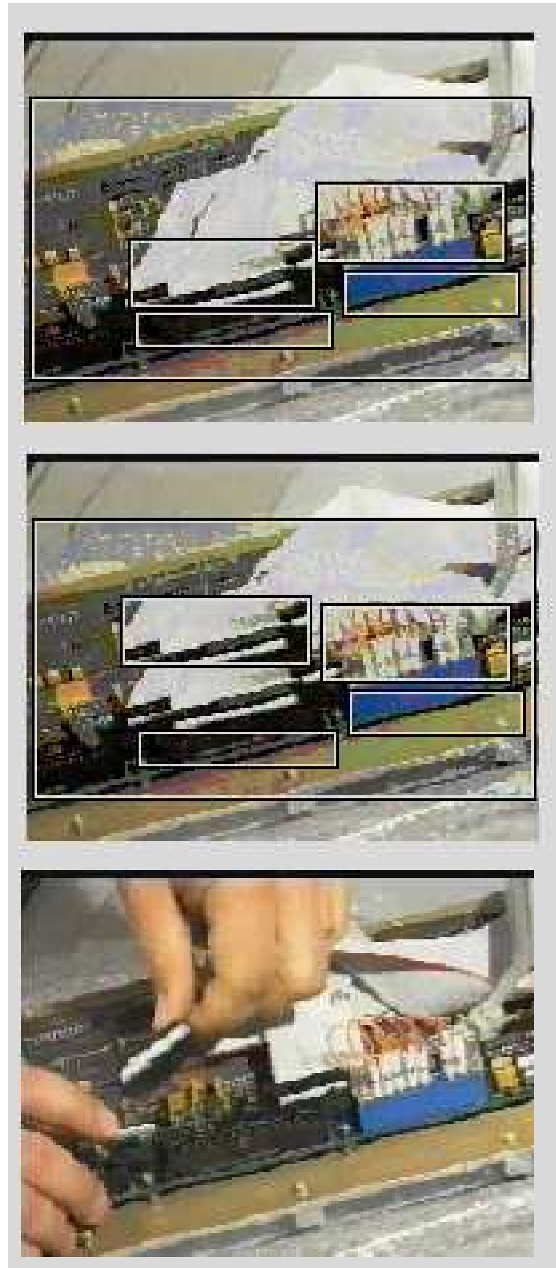
We provide a couple of conveniences that allow re-use of the annotations already described. First, as a result of the visual parsing of the annotations, the annotation graphics are grouped in the graphical editor, so a set of annotations can be moved as a unit from one frame to the next. The result is likely to be mostly correct, and incorrect annotations can be reconnected.

Second, objects in the new frame can be designated as "replacements" for objects in the original frame that appear as arguments to the procedure being defined, and the parts of the new object are made to correspond to the parts of the old object accordingly. The annotations need not be complete for each frame; only those annotations that are relevant for describing each procedure step need be made.

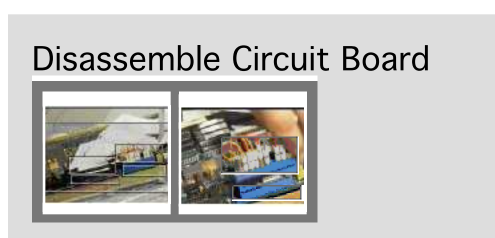
Using the procedure on a new example

Once a procedure has been recorded, it appears in Mondrian's palette in the same form as Mondrian's built-in operations: as a domino of before and after pictures representing the operation. The before and after pictures are miniaturized frames of the initial and final states of the procedure.

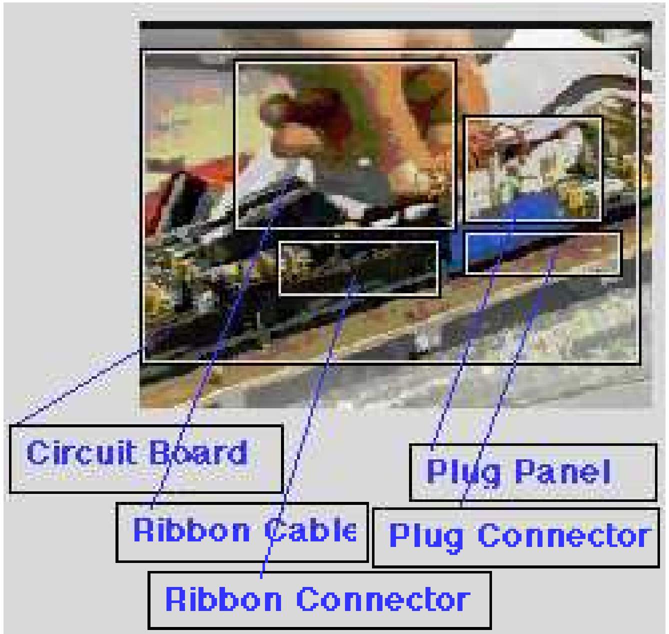
The newly defined operation can then be applied to a new example. Assume we have recorded a two-step procedure that disassembles the ribbon cable assembly and the plug panel assembly. First, we choose a video frame representing the new example. Then, we outline the parts and put annotation labels that represent the input to the procedure.



Initial state (top),
after Disassemble operation (middle)
and final state (bottom)



Domino for the new user-defined operation

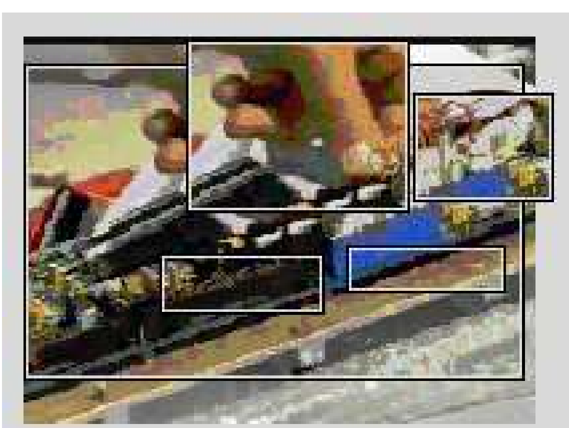


Annotated new example

Now, clicking on the domino for the Disassemble Circuit Board procedure results in performing the disassembly of both parts in the new example.

Mondrian uses an incremental form of explanation-based generalization

The learning method employed by Mondrian had its origin with Tinker (Lieberman 93c) and is similar to the example-based learning technique called *explanation-based generalization* (Mitchell 83). The major difference is the order of "explanations", which is of little theoretical import, but makes a major impact on the usability of this learning technique.



Result of the Disassembly procedure on a new example

Classical explanation-based generalization is a "batch" operation in that the explanation is presented all at once, together with the example. The generalization algorithm produces a generalization from the example and the explanation, which explains to the system "how" the example exemplifies the concept.

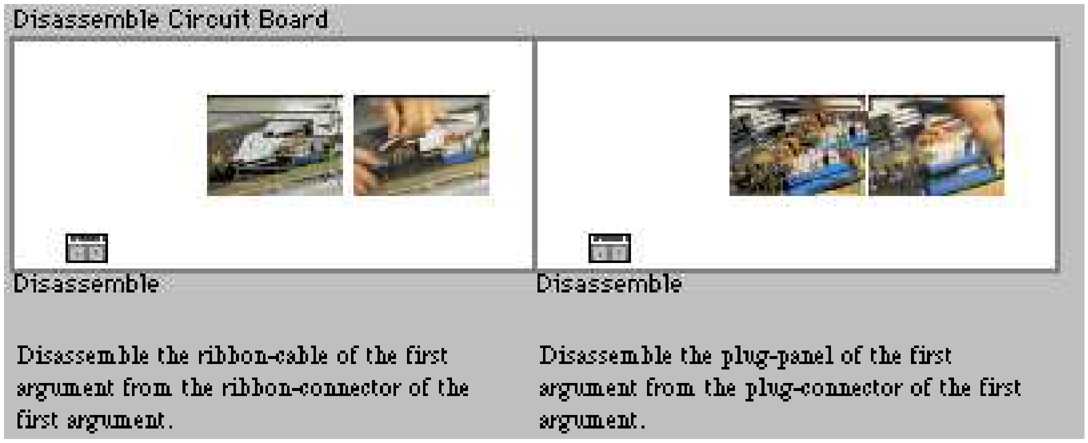
The problem with this as a paradigm for interactive construction of generalizations is that the user is, typically, more confident in his or her choice of examples than in the details of explanation steps, or even in the choice of the exact generalization desired. Feedback from the system on the intermediate steps in the construction of a complex explanation is of vital importance. Especially in domains where the examples are graphical, or can be visualized graphically, immediate graphical feedback can be of immense help to the user in deciding how to explain the example to the system in a way that will cause a desired generalization to emerge. It is not uncommon for the user to realize, in the course of explaining an example, that a related alternative generalization is actually more desirable that the one originally envisaged. The presence or absence of appropriate feedback concerning the relationship between the example and the explanation can make the difference between the usability or impracticality of explanation-based techniques in an interactive interface.

The solution is for the user to provide the "explanation", and the system to construct the generalization, incrementally. We can take advantage of the inherent feedback loop of an interactive interface. At each step, the visible state of the screen indicates the current state of the example, and the explanation is provided by invoking one of the interactive operations, by selecting a menu item or icon. Feedback is generated by the redisplay of the screen, indicating the new state of the example.

The programming by demonstration techniques are also related to case-based reasoning (Schank and Riesbeck 91). Case-based reasoning does not require that the examples be "explained", and learns best when there are a large number of examples. In contrast, demonstrational techniques apply best when there are only a small number of examples, and more guidance from the user is available.

Documenting the procedure

We use a graphical storyboard to represent the procedure in the interface. The storyboard consists of a set of saved pictures of each step of the procedure, together with additional information that describes the step: the name of the operation chosen, a miniature icon for the operation, and the selected arguments are highlighted.



The system also captions each frame of the storyboard with an English description of the step. A simple template-based natural language generator "reads the code" to the user by constructing sentences from templates associated with each function and each kind of object.

The system can also, optionally, speak the description using a text-to-speech generator. It is useful to have the system use speech to give the user feedback after each operation, so that the user can verify that the system actually is understanding the intended description of each step. Speech is a good modality for providing this feedback, because it does not interfere with the hand-eye coordination of operating the graphical editor.

The storyboard provides a form of automatically generated documentation for the procedure being described. Although this documentation is not equal to the quality of documentation produced by a competent technical documentation staff, it is cheap to produce, and guaranteed not to omit crucial steps.

The role of user interface in machine learning

Machine learning has made great strides (Carbonell 1990), but the dream of a learning machine as a useful intelligent assistant to a human being still seems to remain out of reach. My conjecture for what has held up machine learning is that the field has placed an inordinate amount of emphasis on the technical details of algorithms for the computer's inference processes during learning, be they symbolic or connectionist.

In my view, there has not been enough consideration of the interface issues that arise from interaction between a human user and a computer system with learning capabilities. In human education, effective learning occurs not solely as a result of an optimal presentation method on the part of the teacher, or learning method on the part of the student, but upon an effective teacher-student interaction. A variety of particular presentation strategies might be effective, providing that both the teacher and the student find themselves able to communicate effectively with them, including feedback to verify what has been learned.

Thus the machine learning problem is really one of interaction. The key is to establish an interaction language for which the human teacher finds it easy to convey notions of interest, and at the same time, for which the computer as student is capable of learning the appropriate inferences. The "user-friendliness" of the interaction for the teacher is as important as the logical power of the inference method employed by the student.

There is a trade-off that occurs. More powerful inference mechanisms allow the teacher to present ideas of greater complexity with less input, but at the price that it may be harder for the teacher to understand just how to convey a desired generalization, and it becomes harder for the system to present, in a clear and understandable manner, what it has learned and why.

The human interface questions that these issues raise have largely been ignored in the AI literature. Input to learning systems has been largely through assertions coded in special-purpose languages, or sets of coded examples presented in a "batch" to the system. Perhaps the only exception in the AI literature has been in the domain of Intelligent Tutoring Systems (Sleeman and Brown 82), whose audience of beginning users demands special attention to usability at the outset.

Direct-manipulation graphical interfaces have revolutionized human-computer interfaces to personal computer applications. Yet machine learning AI research has been largely aloof from this revolution, taking a "we'll leave interface issues until later" attitude, to concentrate on formalization of generalization methods. The time has now come to couple advances in machine learning with modern, direct-manipulation graphical interfaces. The inability of machine learning researchers to produce learning systems with interfaces that meet contemporary standards blocks acceptance by a user community and integration into contexts where they will receive practical usage.

The interface community itself is reaching a crisis point. "Feature wars" between software vendors produce ever-more complex interfaces, but that still lack features considered essential by some users. The ability for a user to directly teach the system operations of interest permits an interface to remain uncomplicated for beginning users, and to adapt as the user's expertise grows.

Direct-manipulation interfaces pose special challenges for learning systems. Interpretation of the user's point-and-click behavior is the primary one. A point-and-click interaction should be interpreted by the learning system as the equivalent of an assertion in the world of semantic interest. The system must be able to establish a mapping (which may vary according to context) between the manipulable graphical objects and the semantic objects of the domain. Similarly, a mapping easily understandable by the user must be in effect when the system presents feedback about the result of its learning operations.

Related work

Many projects described in the book on Programming by Demonstration (Cypher, ed. 93) influenced this work, though none consider video as a source. Marc Davis' Media Streams (Davis 93) is a system for doing interactive iconic annotation of video sequences. It produces representations in a frame language, but includes descriptive information only and has no demonstrative component. Interactive visual parsing techniques were introduced in (Lakin 86), and (Weitzman and Wittenburg 93) provides a recent example. APEX (Feiner and McKeown 90) use AI techniques to automatically generate text and graphic documentation of technical procedures.

There has been much work in developing direct-manipulation interfaces to be used in conventional knowledge acquisition (McGraw and Harbison-Briggs 89). Most projects of this sort are box-and-arrow browsers and editors for the knowledge structures produced in a conventional knowledge engineering methodology. They don't use video of the target procedure as a source, and visual representations of concrete examples of procedures generally do not play a significant role.

Conclusion

We have shown how graphical annotation and programming by demonstration techniques can be used to provide a tool for constructing formal descriptions of a procedure from a real-world video of the procedure. In the near term future, we will work on better tools for segmenting the video and annotating other kinds of relationships other than part/whole relations. We could also streamline the process of going from the annotation of one set of frames to the next. More effort could also build up a more complete library of actions, and more flexible tools for the user to introduce new action classes and object states. We could also introduce more heuristics for interpreting action sequences. In the long term, incorporation of computer vision algorithms and perhaps, speech recognition of a simultaneous narration of the procedure, could lead to a highly automated video annotation process.

Acknowledgments

Major support for this work comes from research grants from Alenia Corp., Apple Computer, ARPA/JNIDS and the National Science Foundation. The research was also sponsored in part by grants from Digital Equipment Corp., HP, and NYNEX.

References

Apple Computer, *Inside Macintosh: QuickTime Components*, Addison Wesley/Apple, 1993.

Barwise, Jon and John Perry, *Situations and Attitudes*, MIT Press, 1983.

Cypher, Allen, ed. *Watch What I Do: Programming by Demonstration*, MIT Press, 1993.

Davis, Marc, Media Streams, IEEE Symposium on Visual Languages, Bergen, Norway, August 1993.

Feiner, S. and K. McKeown, Coordinating Text and Graphics in Explanation Generation, Proceedings of AAAI-90, Boston, July 1990.

Lakin, Fred, Spatial Parsing for Visual Languages, in *Visual Languages*, S.K. Chang, T. Ichikawa and P. Ligomenedes, eds., Plenum Press, 1986

Lieberman, Henry and Carl Hewitt, A Session with Tinker, Lisp Conference, Stanford, California, August 1980.

Lieberman, Henry, Capturing Design Expertise by Example, in East-West Conference on Human-Computer Interaction, St. Petersburg, Russia, August 1992.

Lieberman, Henry, Mondrian: A Teachable Graphical Editor, in (Cypher, ed. 93).

Lieberman, Henry, Graphical Annotation as a Visual Language for Specifying Generalization Relations, IEEE Symposium on Visual Languages, Bergen, Norway, August 1993.

Lieberman, Henry, Tinker: A Programming by Demonstration System for Beginning Programmers, in [Cypher, ed. 93].

McGraw and Harbison-Briggs, *Knowledge Acquisition: Principles and Guidelines*, Prentice-Hall, Englewood Cliffs, NJ 1989.

Mitchell, T. M., R. Keller, and S. T. Kedar-Cabelli Explanation-Based Generalization: A Unifying View, *Machine Learning* 1(1)47-80, 1986.

Mitchell, T. M., S. Mahadevan, and L. I. Steinberg LEAP: A Learning Apprentice for VLSI Design., In Kodrotoff, Y. and R. S. Michaliski (eds.) Machine Learning: An Artificial Intelligence Approach, Vol. III, pages 271-301, Morgan Kaufmann.

Schank, Roger and Christopher Riesbeck, *Inside Case-Based Reasoning*, Ablex, 1991.

Sleeman, D., Brown, J.S., eds. *Intelligent Tutoring Systems*, Academic Press, New York 1982.

Weitzman, Louis, and Wittenburg, Kent, Relational Grammars for Interactive Design, 1993 IEEE Workshop on Visual Languages, Bergen, Norway.