PML: Representing Procedural Domains for Multimedia Presentations

Technical Report GIT-GVU-98-20
August 1998

Ashwin Ram[1], Richard Catrambone[2], Mark J. Guzdial[1],
Colleen M. Kehoe[1], D. Scott McCrickard[1], John T. Stasko[1]

[1]College of Computing
[2]School of Psychology
Graphics, Visualization and Usability Center
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract

A central issue in the development of multimedia systems is the presentation of the information to the user of the system and how to best represent that information to the designer of the system. Typically, the designers create a system in which content and presentation are inseparably linked; specific presentations and navigational aids are chosen for each piece of content and hard-coded into the system. We argue that the representation of content should be decoupled from the design of the presentation and navigational structure, both to facilitate modular system design and to permit the construction of dynamic multimedia systems that can determine appropriate presentations in a given situation on the fly. We propose a new markup language called PML (Procedural Markup Language) which allows the content to be represented in a flexible manner by specifying the knowledge structures, the underlying physical media, and the relationships between them using cognitive media roles. The PML description can then be translated into different presentations depending on such factors as the context, goals, presentation preferences, and expertise of the user.

Keywords

multimedia, representation, learning, training, cognitive media, design, XML

1. Introduction

Suppose that you are a homeowner faced with the problem of a leaky faucet. If you are reasonably comfortable with home repair, you might be able to just plunge on in and make the repair. But if you are not familiar with that kind of repair, you might seek help, perhaps using a book or even one of the new multimedia guides to home repair for your PC. Will the book present the information you need at the level you need it? Maybe you know home repair well in general, but don't know faucets. Then you probably want a book that presents the key steps, but without a lot of detail. But if you are a novice at home repair, you probably need a lot of examples (with photos and diagrams) and detailed descriptions. In the case of both the book and the multimedia guide, you may encounter the problem of reading the book or the PC screen while your hands have tools in them and you are in the midst of the repair.

This hypothetical scenario is the focus of our research. How can information be presented, dynamically, to meet the needs and prior knowledge of the learner? How should content be encoded so that the developer can present information tuned to the audience and perhaps even presented in the medium of choice?

One of the central issues in the development of multimedia systems, whether on the Web or as a standalone system, is the representation of the content, that is, the information that is to be displayed to the user of the system. For example, an educational system that teaches chemistry must contain information about atoms, molecules, and gas laws; a training system for computer technicians must contain information about memory chips and buses; and a Web site for amateur home repair must contain information about kitchens, showers, and faucets. In designing such systems, however, one must attend not only to the knowledge to be represented but to how that information is to be presented. Is the system's information about showers to be displayed in graphical form? Is information about installing memory chips to be displayed as an itemized list of textual imperatives? Is information about gas laws to be displayed using an animated video clip showing the movement of molecules as a gas is compressed? Typically, the system designer must consider both content and presentation and, in doing so, create a system in which the two are inseparably linked; a specific presentation is chosen for each piece of information, perhaps combining several available media such as text, pictures, animations, and sound, and hard-coded into the system so that it is available for presentation in exactly the form that the system designer intended. Specific navigational aids are also chosen and hard-coded into the system so that a predetermined hypermedia structure is encoded and available to the user.

We argue that *knowledge representation* and *presentation design* should be treated as separate activities. The knowledge engineer or content designer should focus on the knowledge that is being represented: what is known about molecules, buses, and faucets, how this knowledge is structured, and how it might be represented using basic media elements. The presentation designer should focus on the multimedia presentation of this knowledge: how should a diagnostic procedure be displayed to the user? At what level of detail, and for what level of expertise should it be presented? What should it be linked to? What navigational aids should be provided? What issues concerning how people comprehend text, graphics, animations, etc. need to be considered?

Such an approach has several benefits. First, it is easier for domain experts (who may not be presentation experts) to build the knowledge representation without regard to how the information will ultimately be displayed. Second, it is possible to design different presentations for the information based on the user's level of expertise, or on the task that the user is engaged in (for example, learning vs. troubleshooting), or on other factors. Third, this approach permits the development of truly interactive multimedia systems in which the system creates appropriate presentations on-the-fly based on the current interactions and context. Only the knowledge needs to be specified beforehand, but whether a diagnostic procedure, for example, is presented as an itemized list of textual bullets, a graphical flow chart, an animated movie, or some combination thereof can be determined dynamically. If knowledge and presentation were tightly coupled, all these presentations would have to be created manually in advance and stored as alternative depictions of the same information.

Of course, the knowledge representation must ultimately bottom out in media: a textual definition of a gas law, a photograph of a faucet, or a schematic of a VLSI chip. Thus, it is important to provide a principled means of coupling the knowledge representation structures with the underlying media, but in a manner that provides the flexibility needed for interactive and dynamic presentations. We argue that knowledge structures should organize media according to their *cognitive role* [1]. Consider, for example, a student who is using an educational multimedia system to learn chemistry, or a homeowner who is using a home repair CD-ROM or Web site to help fix a leaky faucet in a bathroom. The user is unlikely to say, "I would like to see some text now" or "I could really use a WAV sound file now." Instead, the user may say, "I could really use an example," leaving it up to the system to determine whether that example is best presented as text, sound, animation, or some combination thereof. In other words, we argue that multimedia content, consisting of *physical media* such as text, sound, video clips, and so on, should be organized and coupled to knowledge structures using *cognitive media roles*, such as definition, example, simulation, worked problem, and so on. A cognitive media role, such as "example," specifies the function that the information plays in the cognitive processes of the user. The user might ask for an example of a faucet or a simulation of molecular forces, which in turn would be displayed using an appropriate combination of physical media as determined statically by the presentation designer and/or dynamically by the system itself.

There has been a significant amount of work attempting to disentangle knowledge representation from presentation in multimedia. Maybury [2] has emphasized this distinction in his work with intelligent multimedia interfaces. Feiner [3] has shown a system that can actually construct multimedia representations on-the-fly, drawing from a knowledge base of content and a separate knowledgebase about representations. Research teams such as the Hyper-G group in Austria [4] have created Web-based applications that allow for separation between representation and presentation (e.g., keeping link information separate from the multimedia document itself). What we add to the existing science is (a) a theory of effective organization for multimedia used for **learning** (specifically, cognitive media roles), and (b) a notation for encoding knowledge such that an effective representation can be generated.

To facilitate the development of a system outlined above, we propose and describe in this article a new notation called Procedural Markup Language (PML). PML is a markup language written in XML that allows the content designer to encode domain knowledge in an intuitive and flexible manner by specifying the knowledge structures, the underlying physical media, and the relationship between them using cognitive media roles. We focus specifically on procedural task domains, in which the primary type of knowledge to be represented concerns the performance of procedures. The highlights of our formalism are:

- Information about a domain (e.g., plumbing) is encoded in *knowledge nodes* that have connections, called *knowledge links,* to other knowledge nodes.

- Information within a particular knowledge node (e.g., information about a faucet) is represented using *physical media clusters* containing media elements such as text, graphics, animations, video clips, and sound files.

- Physical media are organized under knowledge nodes using *cognitive media roles*, such as "definition," "example," etc. Any cognitive media role under a knowledge node could potentially contain one or more different physical media (e.g., an example of a faucet might be represented using some text and a graphic).

- The information contained in the combination of knowledge nodes, knowledge links, physical media clusters, and cognitive media roles forms the raw material that can be used by a presentation system to determine what the user or learner will see and hear, and what navigational connections and devices will be available on the screen. Different presentations may be created from the same underlying representation, depending on various factors such as the expertise of the user in the domain, the information that the user has previously seen, the current goal of the user, and so on.

In this article, we use the home repair domain as our example, and show how PML can be used to represent information about, for example, repairing a leaky faucet. The PML representation is independent of the particular presentation that is ultimately constructed. We show that the same PML representations can be used to create different presentations. Since we have focused primarily on the development of PML, our current implementation the presentation-construction system is fairly simple. Ultimately, we are interested in more sophisticated presentation-construction systems which can dynamically create an appropriate presentation based on the goals or tasks of the user, the user's level of expertise, the context, and other appropriate factors.

2. Technical Details: Knowledge Representation

PML consists of *knowledge nodes* that represent concepts and *knowledge links* that represent relationships between knowledge nodes, similar to the semantic net structures used for knowledge representation in artificial intelligence systems. However, unlike semantic net systems which are used for reasoning, nodes do not contain slot-filler representations of concepts; for multimedia presentations, nodes need to contain media that can be used to create presentations of those concepts for the user. Media are stored in *physical media clusters* that contain text, pictures, sounds, video clips, etc., that are the basic elements describing concepts. Media clusters are organized under knowledge nodes using *cognitive media roles* that represent the cognitive role (e.g., example, definition) played by the media elements in describing the concepts in the knowledge nodes. A cognitive media role, such as an example, may have one or more physical media clusters associated with it, where each cluster represents a different example. This structure is summarized in Figure 1.

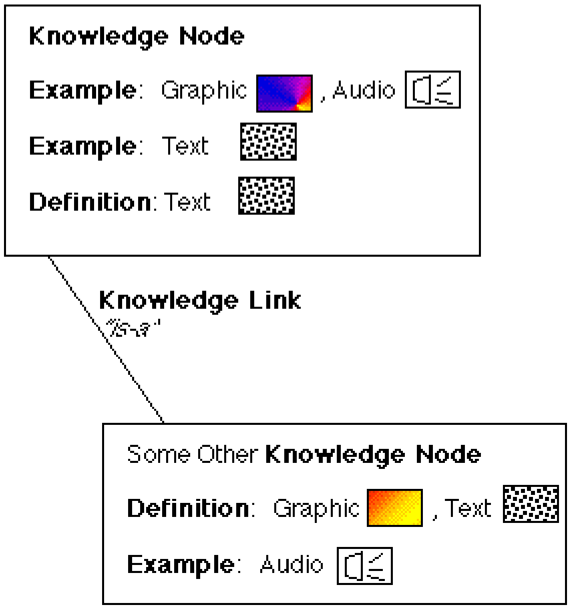


Figure 1: General Structure of Knowledge in PML Representations

Before describing the markup language itself, it is instructive to look at the representational structures that the language must encode. Let us briefly discuss the necessary nodes and links.

Knowledge Nodes

A knowledge node represents a concept that the system knows about. Information about the concept is represented using media clusters while relationships between concepts are represented using knowledge links. Following basic ontological principles that are commonly used in representations systems in artificial intelligence and cognitive science, we divide the entities being represented into *things*, *states*, and *procedures*(see Table 1). Based on our experience with several different domains, this ontology appears to sufficient to capture the distinctions necessary to represent our target domains where the knowledge being represented is mainly procedural. In general, though, the robustness of the language can only be determined by representing a large number of domains using this formalism.

Table 1: Knowledge Nodes

Name	*Description*	*Examples*
Thing	Represents a system, physical object, part, or substance in its normal state. Things may be composed of other things.	Hot water system (system). Faucet (physical object). Washer (part). Water (substance).
State	Each thing has one or more states that it can be in. A thing's normal state is the usual operational state of that thing; other states represent problem conditions that may need repair. Usually only problem states are represented explicitly; the main representation of a thing is assumed to represent its normal state.	Pilot light off. Faucet leaky. Washer worn out. Water is brown.
Procedure	Represents a sequence of actions carried out by the user that operate on a thing in some manner. The actions that comprise a procedure may themselves be procedures; ultimately, this bottoms out when the "primitive" action is operationalized and can be directly carried out by the user.	Lighting a pilot lamp. Replacing a leaky faucet. Installing a shower.

Knowledge Links

Knowledge nodes may be linked to other knowledge nodes using knowledge links that represent conceptual relationships between those knowledge nodes (see Table 2). Note that knowledge links are strongly typed; for example, the **precondition** link always connects states to procedures. The order in which multiple links of the same type are listed under any givennode is not significant, with the exception of **steps** which are listed in the order in which they should be carried out. Knowledge links may be traversed in either direction by the system, although each link has an explicit source and destination endpoint. The reverse links are also listed in Table 2. These reverse links are managed by the system and not manually created by the user. Our formalism provides the following set of knowledge links; as before, determining the sufficiency of this set is an empirical question, although this set has been adequate for several domains that we have investigated.

Table 2: Knowledge links

Name	*Description*	*Examples*
Is-a	Thing **is-a** Thing Represents the broader category of a thing, or (the other direction) the particular types of a thing. The reverse link is **subtype**.	Single-lever faucet **is-a** Faucet Faucet **subtype** single-lever faucet Pilot light **is-a** Ignition device
Is-a	Procedure **is-a** Procedure Represents the broader category of a procedure, or (the other direction) particular styles of a procedure. The reverse link is **subtype**.	Plunge-drain **is-a** unclog-drain Unclog-drain **subtype** plunge-drain
Has-a	Thing **has-a** Thing Represents a subsystem of a system or a part of a physical object. Only things can have parts, which are other things. The reverse link is **part-of**.	Water heater **has-a** Pilot light Pilot light **part-of** water heater Faucet **has-a** Washer Hot water system **has-a** Shutoff valve Hot water system **has-a** Stepped pipes Hot water system **has-a** Water heater
Connects-to	Thing **connects-to** Thing Represents contiguous or connecting pieces of an overall physical system. The overall physical system, represented as a thing, would have **has-a** links to the individual things comprising it as well. The reverse link is **connects-to**.	Shutoff valve **connects-to** Stepped pipes **connects-to** Water heater **connects-to** Hot water supply line
Steps	Procedure **steps** Procedure	Replace washer **steps** (Unscrew nut;

	Represents the substeps of a procedure, that is, steps that represent the procedure in more detail (these steps may, in turn, be further broken down into substeps). An experienced user may choose not to see this level of detail. There is an implied ordering of the steps of a procedure. The reverse link is **step-of**.	Remove washer; Insert new washer; Replace nut) Unscrew nut **step-of** Replace washer
Problem-state	Thing **problem-state** State Links things to the problem states that those things can be in. A problem state is an abnormal state that requires repair. A thing may have multiple problem states. The reverse link is **problem-state-of**.	Water heater **problem-state** Pilot light off Pilot light off **problem-state-of** water heater Faucet **problem-state** Faucet leaky Facuet leaky
Repair-procedure	State **repair-procedure** Procedure Links a problem state to a procedure that, if successfully completed, repairs the problem and returns the thing to its normal state. A problem state may have multiple repair procedures. An installation procedure is also represented as a repair procedure. The reverse link is **repair-procedure-for**.	Pilot light off **repair-procedure** Relight pilot light Relight pilot light **repair-procedure-for** pilot light off Faucet leaky **repair-procedure** Repair leaky faucet No bathtub in bathroom **repair-procedure** Install bathtub
Outcome	Procedure **outcome** State Links a procedure, which could be an entire procedure or an individual primitive step within a procedure, to the states that result from carrying out that procedure. The state may be presented to the user as evidence that the procedure was successfully carried out. A procedure may have more than one possible outcome. The reverse link is **result-of**.	Tighten washer **outcome** Water does not leak through Water does not leak through **result-of** tighten washer
Outcome	State **outcome** State When there is no intentional intervening action, an outcome link may link a state directly to another state that may result. The reverse link is **results-from**.	Faucet leaky **outcome** Basement wet Basement-wet **results-from** faucet leaky
Precondition	State **precondition** Procedure Links the preconditions or enabling conditions of an action to	Pilot light off **precondition** Open water heater access

	the node that represents that action. The reverse link is **requires**.	panel Open water heater access panel **requires** pilot light off
Uses	Procedure **uses** Thing Links procedures to tools, instruments, and other objects that are used in that procedure. The rese link is **used-in**.	Shut off water **uses** Monkey wrench Monkey wrench **used-in** shut off water
Related-to	Links any node to any other node that may contain related or relevant information. Used (sparingly!) to represent any relationships not specifically captured by existing link types. The reverse link is **related-to**.	Hot water system **related-to** Heating system Install faucet **related-to** install shower Washer **related-to** Repair leaky faucet

Physical Media Clusters

A physical media cluster (or simply media cluster) contains the actual information about a knowledge node that the system can display to the user. We will see below that all media are stored in separate files, referenced via the `MEDIA` tag, with the exception of text which may be included in-line in the PML document for authoring convenience. A media cluster may contain more than one type of physical media (text, video, etc.). A knowledge node may contain one or more media clusters; these are organized using cognitive media roles that provide the connections between the knowledge structures and the media clusters.

Cognitive Media Roles

The media clusters within a knowledge node are organized in terms of the cognitive roles they play in the problem-solving task in which the user is engaged. For example, a particular mixed-media text-and-pictures description of a faucet may serve as an "example" of a single-lever faucet; in this case, the knowledge node for "single-lever faucet" will contain an "example" role which contains a media cluster that represents that text-and-pictures description. Our formalism provides the following cognitive media roles [1].

Table 3: Cognitive media roles

Name	*Description*	*Example*
Name/ Title	The name of the item being represented in the knowledge node. Usually one or a few words of text.	"Pilot light."
Definition/ Description	The definition of the concept being represented in the knowledge node or, more informally, its description. Usually a textual description accompanied by diagrams.	"A pilot light is a small gas flame that is continually burning. It is used to light the furnace when ... ..." [Schematic of pilot light]
Example	An example of the concept being represented in the knowledge node.	[Photograph of an actual pilot light] "This is an example of a pilot light. In this design, the small lever to the left [pointer to picture] is used to ..."
Counter-example	An example of an alternative design, an alternative state, etc.	[Photograph of a flint-based lighting system.] "An alternative to the pilot light is the ..."
Justification	An explanation of a step being carried out in a procedure, or an explanation of the functional role of a thing that is part of a larger thing.	"You want to turn off the water at the supply first because ..." "Faucets have O-rings because..."

Example

To illustrate how these nodes and links fit together to provide an overall representation of the domain of interest, consider a snippet of the representation of faucets from the home repair domain, shown in Figure 2 in pictorial form.

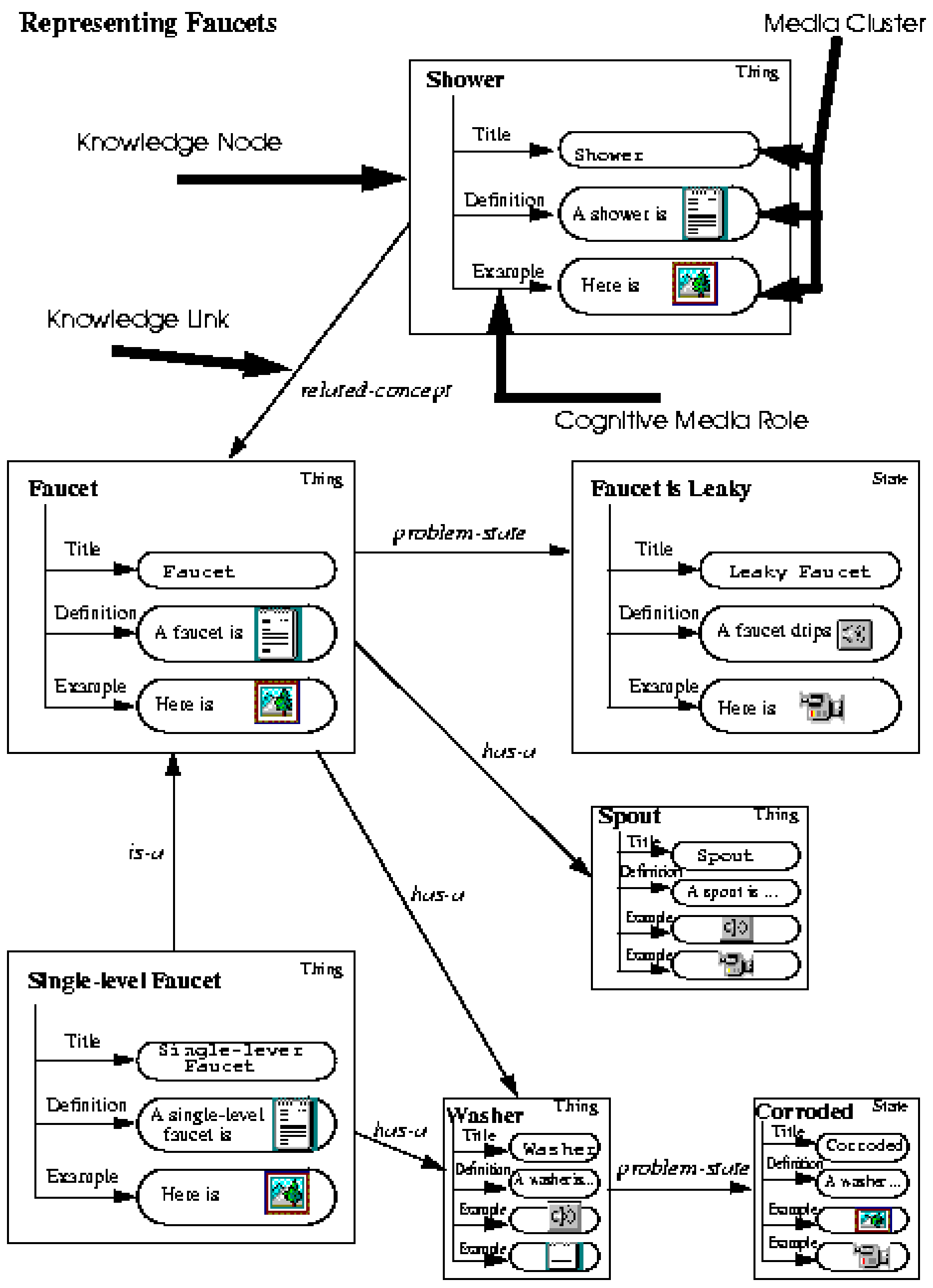


Figure 2: Pictorial form of a PML knowledge structure about faucets.

3. Technical Details: Procedural Markup Language

The previous section described our knowledge representation framework. Briefly, knowledge is organized into nodes (concepts) that are linked by relationships. Each node organizes physical media into clusters corresponding to roles for the media. This was summarized in Figure 1. In this section, we describe the notation that we use for articulating this representation.

We have developed a language called PML that allows authors to encode our knowledge representation in a set of files. Authoring in PML is analogous to authoring in HTML or other markup languages, but with the crucial difference that the author focuses on representing information about the domain and not primarily on information about presentation. That is, the two types of information are decoupled. For example, if one were to create a Web page in HTML describing how to install the latest release of a piece of software, one might do this as follows:

```
To install this release, perform the following steps: <BR>
<OL>
<LI> Download the file.
<LI> Unstuff the file.
<LI> Double-click on the installer icon.
</OL>
```

Notice that the procedure is described using a series of steps, but in order to state the steps one has to choose a particular physical representation (here, a numbered list of items). Using PML, however, one would specify the steps independent of the presentation:

```
<PROCEDURE ID="install">
<TITLE>Installing the software</TITLE>
<DESCRIPTION>To install this release, perform the following
   steps:</DESCRIPTION>
<LINK TYPE="steps">
   <TARGET ID="download"/>      <TARGET ID="unstuff"/>
   <TARGET ID="execute"/>
</LINK>
</PROCEDURE>
```

This example presents a *procedure* knowledge node with two cognitive media roles: *title* and *description*. The knowledge node has *step* knowledge links to separate "download," "unstuff," and "execute" procedure nodes that are represented using PML as well. Note that the PML does not specify whether to display the three steps as a numbered list, a flow chart, or as three separate pages with navigational arrows between them; that decision can be made independently and, if desired, dynamically (limited only by the physical media provided). Note also that the PML representation allows additional types of information to be represented, such as the preconditions and outcomes of the procedure, things that might go wrong and how to recover from them, justifications for the procedures, and so on. These could be hard-coded into the HTML representation too, but again the presentation would be static. Finally, the PML representation allows procedures and subprocedures to be represented hierarchically; the level of detail that is actually presented can be determined dynamically and should be dependent on factors such as the expertise of the user.

PML is written in Extensible Markup Language (XML) [5], a language for describing other markup languages. XML is a simplified version of Standard Generalized Markup Language (SGML), an international standard for creating structured documents. A markup language is essentially a set of tags that an author uses to describe parts of a document and a document that uses these tags is one kind of structured document. Currently, the most well-known markup language is HTML which contains tags like <TITLE>, <H1>, <IMG>, etc. While these tags are useful for describing basic document structure, they do not describe the content of a document very well. More powerful and most likely domain-specific markup languages are needed for this purpose and XML was developed as a common way to define these different markup languages. XML, however, has utility far beyond the World Wide Web; it can be thought of as a platform-independent way to represent knowledge in a machine-readable format. XML has already been used to specify markup languages for dozens of applications ranging from chemistry to electronic commerce [6]. Having a common way to specify these markup languages allows tools to be built that can work with any of the languages specified in XML. For example, we have developed a PML-to-HTML translator that uses a PML parser. This parser is generic, however, understanding XML and therefore any markup language specified using XML.

The notation used in XML descriptions is fairly standard (see [5] for details). Each ELEMENT statement is a production rule with the first item being the left-hand side of the rule and the second item (in parentheses) being the right-hand side. Every element corresponds to a tag in the markup language. A vertical bar indicates a choice and a comma indicates a sequence. The plus sign stands for "one or more" and the asterisk stands for "zero or more." An ATTLIST statement lists the attributes for a particular element (i.e. tag). It specifies the type of the attribute and whether it is required (REQUIRED) or optional (IMPLIED). The complete specification of our PML language is given in Table 4. Appendix A contains an annotated example PML representation of a snippet of an everyday procedural domain: baking a cake.

Table 4: Specification of PML

```
 <!----  Things  ---->

<!ELEMENT thing   (title, (author | description | justification | link |
appspecific | example | counterexample )*) >
<!ATTLIST thing           id   ID   #REQUIRED>

<!----  States  ---->
```

```
<!ELEMENT state   (title, (author | description | justification | link |
appspecific | example | counterexample )*) >
<!ATTLIST state           id   ID   #REQUIRED>

<!----  Procedures  ---->

<!ELEMENT procedure   (title, (author | description | justification |
link | appspecific | example | counterexample )*) >
<!ATTLIST procedure       id   ID   #REQUIRED>

<!----  Cognitive  Media  Types  &  Identifying  Information  ---->

<!ELEMENT title          (#PCDATA | media)* >
<!ELEMENT author         (#PCDATA | media)* >
<!ELEMENT description    (#PCDATA | media)* >
<!ELEMENT justification  (#PCDATA | media)* >
<!ELEMENT example        (#PCDATA | media)* >
<!ELEMENT counterexample (#PCDATA | media)* >
<!ATTLIST description        type   CDATA    #IMPLIED>
<!ATTLIST justification      type   CDATA    #IMPLIED>
<!ATTLIST example            type   CDATA    #IMPLIED>
<!ATTLIST counterexample     type   CDATA    #IMPLIED>

<!----  Media  ---->

<!ELEMENT media  #PCDATA
<!ATTLIST media            src      CDATA    #REQUIRED
                           caption  CDATA    #IMPLIED>

<!----  Links  &  Targets  ---->

<!ELEMENT link   (target+)>
<!ATTLIST link           type   (uses | is-a | has-a | connects-to |
related-to | steps | precondition | outcome | problem-state | repair-
procedure) #REQUIRED>

<!ELEMENT target  EMPTY>
<!ATTLIST target           id       IDREF    #REQUIRED>

<!----  Application-Specific  Key/Value  Pairs  ---->

<!ELEMENT appspecific   EMPTY>
<!ATTLIST appspecific     key       CDATA    #REQUIRED
                          value     CDATA    #REQUIRED>
```

Authoring tools

In order to facilitate the authoring of PML documents, we have developed a graphical editing tool called tkPML that can be used to create the knowledge node/link networks graphically (see Figure 3). As might be expected, textual hand-authoring of PML structures can be tedious and mistake-prone. The tkPML graph creation interface replaces the task of entering the node and link information by hand with a point-and-click interface with form fill-in for the required text entry. We expect that this tool will help designers to better establish and maintain mental models of their PML representations.

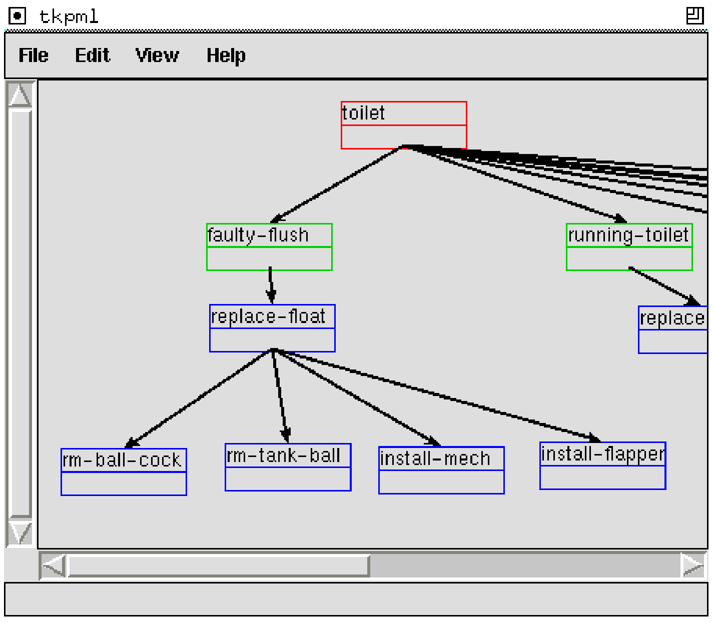
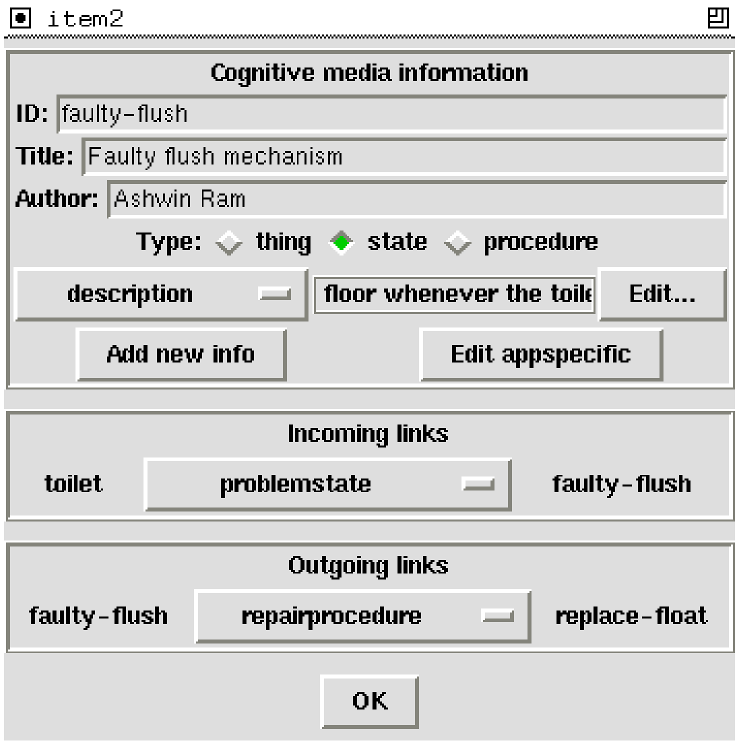


Figure 3: A screenshot of a tkPML session. Nodes are created by double clicking on the background and are moved by dragging on the top "title" portion of the node. Links are created by dragging from the bottom "link" portion of one node to another. Double clicking on an existing node pops up a node information screen (shown in Figure 4) that allows a designer to edit information about the node.

In tkPML, nodes and links can be created and positioned with simple mouse actions, and the layout can be seen at various levels of detail to obtain an overview of large knowledge structures as well as a more informative view of a smaller number of nodes. Each node can be expanded to view and change the knowledge contained in it (see Figure 4).

The tkPML tool saves a PML description file in an augmented PML format file. The one addition is simply node location, which is not part of the standard PML definition and is saved as a comment. Thus, designers can edit the saved files by hand or run presentation interpreters on the files without modification. In addition, tkPML can import files written in PML, even files that were not created using tkPML. For such files, tkPML uses a simple graph layout algorithm to generate an initial display.


item2

Cognitive media information

ID: faulty-flush

Title: Faulty flush mechanism

Author: Ashwin Ram

Type: thing [✓ state] procedure

description | floor whenever the toile | Edit...

Add new info | Edit appspecific

Incoming links

toilet | problemstate | faulty-flush

Outgoing links

faulty-flush | repairprocedure | replace-float

OK

Figure 4: View of a knowledge node in the tkPML tool. In the top area are indicated the default cognitive media roles and the capability to add others. The lower area shows the incoming and outgoing knowledge links.

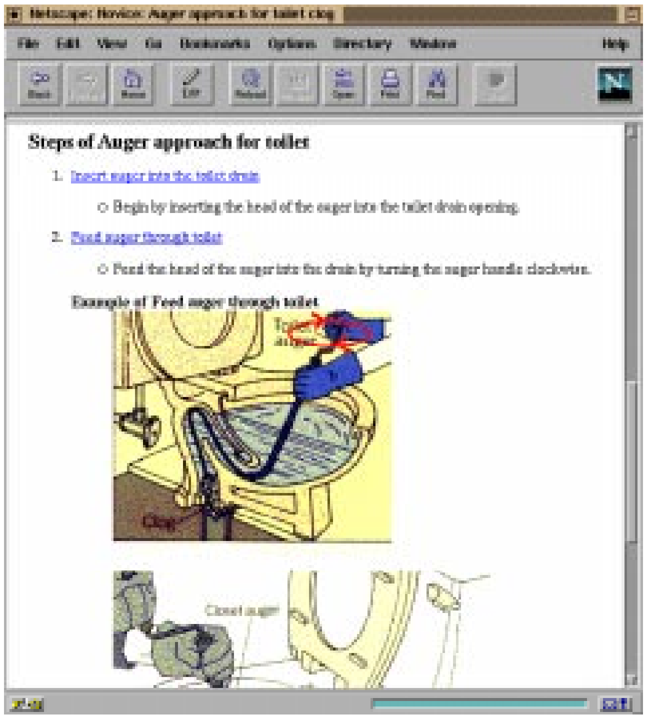
As the name suggests, tkPML is written in Tcl/Tk, a platform-independent graphical scripting language that can run on Unix machines, PCs, Macintoshes, and within browsers over the World Wide Web. This allows PML representations to be exchanged and edited by users on different platforms and even published on the Web.

Presentation tools

In order to create a presentation based on a PML document, one may develop a PML interpreter-generator that can interpret PML descriptions, retrieve the appropriate media in those descriptions, and create a hyperlinked presentation based on the situation and needs of the user. For example, if a novice user is considering whether to call a plumber to repair a leaky shower, the system need not display all the details of the repair procedure but instead may choose to summarize the time, expertise, and tools necessary to perform the procedure. If the same user has previously repaired a leaky faucet, the system may display an overview of the repair procedure (the top-level steps) and provide links to the substeps that are different from the previous procedure with which the user has experience. Truly interactive and dynamic multimedia systems will need such capabilities, and PML is designed to support the development of such systems. The PML interpreter-generator would need to be based on principles of instructional and interaction design. PML is designed to support experimentation with such principles.

Alternatively and more simply, one may develop a PML interpreter that can construct a small number of predetermined presentations (such as one that always displays substeps in detail and another which always displays substeps as titles with links to the details) and use a simple heuristic to decide which presentation to use. We have chosen this approach for our initial implementation in order to test PML representations. Specifically, we have developed a PML-to-HTML translator that can create presentations based on simple, predetermined presentation rules (see Figure 5). This allows us to experiment with PML and gain knowledge that will facilitate later development of a fully dynamic presentation system.

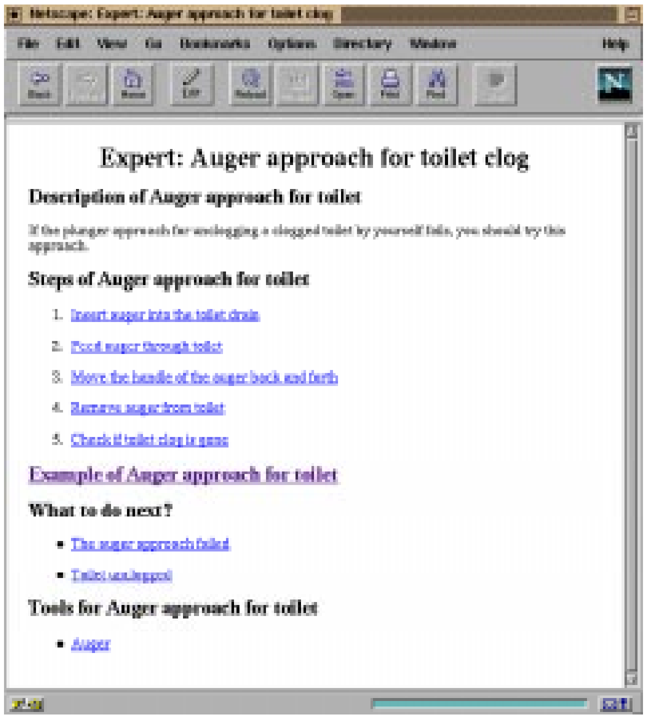
Both of the presentations in Figure 5 are designed to help the user unplug a toilet drain using an auger. The left presentation is aimed at a novice, and the right presentation is aimed at an expert. For the expert, the basic steps are presented with minimal explanation, but with links that lead to more information, if desired. For the novice, more introductory information is provided (not shown) such as what an auger is, and each step is expanded with its explanation and any examples available for the step. Both of these presentations assume a web browser on a personal computer as the target platform. If the target were for, say, a handheld personal computer (which might be more useful when working in the bathroom) or even an audio presentation, a different structure should be generated (e.g., the handheld should not have a long scrolling presentation, as does the novice presentation depicted in Figure 5).


Netscape: Novice: Auger approach for toilet clog
File Edit View Go Bookmarks Options Directory Window Help
Clog
Closet auger

Steps of Auger approach for toilet

1. Insert auger into the toilet drain
   - Begin by inserting the head of the auger into the toilet drain opening.
2. Feed auger through toilet
   - Feed the head of the auger into the drain by turning the auger handle clockwise.

Example of Feed auger through toilet


Netscape: Expert: Auger approach for toilet clog
File Edit View Go Bookmarks Options Directory Window Help

Expert: Auger approach for toilet clog

Description of Auger approach for toilet

If the plunger approach for unclogging a clogged toilet by yourself fails, you should try this approach.

Steps of Auger approach for toilet

1. Insert auger into the toilet drain
2. Feed auger through toilet
3. Move the handle of the auger back and forth
4. Remove auger from toilet
5. Check if toilet clog is gone

Example of Auger approach for toilet

What to do next?

- The auger approach failed
- Toilet unclogged

Tools for Auger approach for toilet

- Auger

Figure 5: Two different presentations based on the same PML source

4. Conclusions

In this paper we present a new markup language called PML to facilitate the authoring of dynamic multimedia systems for procedural domains. We show how PML can be used to represent domain knowledge (concepts and relationships) independent of presentation issues and how this knowledge can be loosely coupled to presentation media via cognitive media roles. PML involves knowledge nodes connected by knowledge links. The knowledge nodes can contain cognitive media roles holding physical media clusters.

Cognitive media roles have been used succesfully in educational multimedia systems for teaching graph algorithms in an undergraduate computer science course [1,7], and Lewis structures in an undergraduate chemistry course [8]. PML has also been used for other tasks and domains; we are using it to represent cases of object-oriented design and programming [9], and to encode process information about operations in an electronic assembly "Clean Room" [10]. While these earlier systems were not dynamic, they do illustrate the generality and value of the knowledge representation and the notational tools.

In our own research, we are developing PML-based systems to investigate cognitive issues relevant in the design of dynamic multimedia systems. More broadly, in a learning situation, the goals that students bring to the learning task will affect their learning processes and therefore a hypermedia support system for learning should have the capability to adjust itself in response to the user's goals. PML allows us to examine these issues empirically. For example, we would like to conduct systematic experiments that look at what factors actually play a role in the effectiveness of different presentations to the user or learner. For instance, is it really the case that a "high-level" presentation of a procedure for a more knowledgeable person is more effective (measured, perhaps, in how well the person can do the procedure and how long it takes, counting presentation time) than providing him or her with all the details? In order to empirically answer questions such as these, we need a system capable of creating alternative presentations of some underlying information, which is precisely the goal of PML.

Acknowledgements

This research was supported by the Office of Naval Research under contract N00014-95-1-0790. We thank Mimi Recker for her comments on an earlier draft of this paper.

References

[1] M. Recker, A. Ram, T. Shikano, G. Li and J. Stasko, “Cognitive Media Types for Multimedia Information Access,” *Journal of Educational Multimedia and Hypermedia*, Vol. 4, No. 2/3, 1995, pp. 185-210.

[2] M. Maybury, Ed., *Intelligent Multimedia Interfaces*, MIT Press, Cambrudge, MA, 1993.

[3] S. Feiner and K. McKeown, “Automating the Generation of Co-ordinated Multimedia Explanation,” *IEEE Computer*, Vol. 24, No. 10, Oct. 1991, pp. 33-41.

[4] Tomek, I., Maurer, H., and Nasser, M., “Optimal Presentation of Links in Large Hypermedia Systems,” *Proceedings of ED-MEDIA ‘93*, 1993, pp. 511-518. See also http://www.hyperwave.com

[5] T. Bray, J. Paoli, and C.M. Sperberg-McQueen, Eds., “Extensible Markup Language (XML) 1.0,” W3C Recommendation 10-February-1998, http://www.w3.org/TR/REC-xml-19980210.html.

[6] R. Cover, “XML: Proposed Applications and Industry Initiatives,” 1998. http://www.sil.org/sgml/xml.html#applications.

[7] G. Shippey, A. Ram, F. Albrecht, J. Roberts, M. Guzdial, R. Catrambone, M. Byrne, and J. Stasko, “Exploring Interface Options in Multimedia Educational Environments,” *Proceedings of the Second International Conference on the Learning Sciences*, Evanston, IL, 1996.

[8] M. Byrne, M. Guzdial, P. Ram, R. Catrambone, A. Ram, J. Stasko, G. Shippey, and F. Albrecht, “The Role of Student Tasks in Accessing Cognitive Media Types,” *Proceedings of the Second International Conference on the Learning Sciences*, Evanston, IL, 1996.

[9] M. Guzdial, “Technological support for an apprenticeship in object-oriented design and programming,” *Proceedings of the OOPSLA'97 Educators Symposium*, Atlanta, GA, ACM, 1997.

[10] M.T. Realff, T. Hübscher-Younger, et al., “Multimedia Support for Learning Advanced Packaging Manufacturing Practices,” *Proceedings of ECTC'98*, IEEE, 1998.

Appendix A

Table 5: Example PML file

```
<PML> # All PML documents must start with the PML tag

<PROCEDURE id="cake 1">
# This says we're beginning a procedure. We've given it an id of "cake 1."
# This is the name you use to refer to this procedure in other places.

<TITLE>How to Bake a Cake</TITLE>
<AUTHOR>Colleen Kehoe</AUTHOR>
# We define the title of this procedure and the author. Title is required.
# The title may be the same as the id in the procedure tag, if desired.

<DESCRIPTION>
This procedure tells you how to bake your basic cake. It assumes you're at or near sea level. You'll need a
different procedure if you're at a high altitude.
</DESCRIPTION>
# We give a description of the overall procedure.
# Since it is text, we can include it here or reference an external file via a MEDIA tag.

<JUSTIFICATION>
Everybody likes cake.
</JUSTIFICATION>
# We give a justification for this procedure. This is optional.

<APPSPECIFIC key="difficulty" value="easy"/ >
# Here we may associate some application-specific information with this node.
# This may be used for indexing purposes or for deciding how to display this node.

<EXAMPLE>
<MEDIA SRC="cake.gif" CAPTION="Here is a picture of someone baking a cake."/>
<MEDIA SRC="cake.mov" CAPTION="Here is a movie of a baker at work."/>
</EXAMPLE>
# An example containing two physical media files.
# Any number of examples or counterexamples are allowed.

# Now we list all of this links from this node to other nodes in the system.

<LINK type="uses"> #This is a list of the equipment this procedure uses.
<TARGET id="mixer"/> #This is a "thing" node.
</LINK>

<LINK type="problem-state"> # The following nodes are problems.
<TARGET id="cake didn't rise"/> # Each is a "state" node.
<TARGET id="cake burnt"/>
<TARGET id="cake tastes salty"/>
</LINK>

<LINK type="outcome"> # The following node is an outcome.
<TARGET id="cake is done"/> # This is a "state" node.
</LINK>
```

```
<LINK type="steps"> # This is a list of the steps in this procedure.
<TARGET id="mix ingredients"/> # These are "procedure" nodes.
<TARGET id="put in oven"/>
<TARGET id="test for doneness"/>
<TARGET id="cool"/>
</LINK>

<LINK type="related-to"> # Other related nodes.
<TARGET id="baked good"/>
</LINK>

</PROCEDURE>
# This marks the end of this procedure.

<PROCEDURE id="mix ingredients">
# This is the beginning of a new procedure. Notice that this is the first step in the
# procedure we just defined above. Procedures are defined hierarchically.

<TITLE>Mix the Ingredients</TITLE>
<AUTHORColleen Kehoe</AUTHOR>

<DESCRIPTION>
Get all the ingredients together and mix them.
</DESCRIPTION>
# For this application, we've provided two descriptions, both text.

<DESCRIPTION type="high">
You will need: 2 eggs, 2 c. flour, 1/2 c. milk, 3 tbsp. butter, 1 tsp. baking soda, 1/4 tsp. salt, 1/4 c. water,
1c. sugar. Combine the dry ingredients in one bowl. Combine the wet ingredients in another bowl.
Gradually add the dry to the wet, blending with an electric mixer.
</DESCRIPTION>
# Here, we provide a very detailed description.

# As before, we list the links from this node to other nodes in the system.

<LINK type="uses"> # This is a list of the equipment this procedure uses.
<TARGET id="mixer"/> # This is a "thing" node.
</LINK>

</PROCEDURE>

# These are the rest of the steps in the "cake 1" procedure.
# Details are omitted in the interest of space, but they would be similar to the one above.

<PROCEDURE id="put in oven">...</PROCEDURE>

<PROCEDURE id="test for doneness">...</PROCEDURE>

<PROCEDURE id="cool">...</PROCEDURE>

<THING id="baked good">
# Now we define a "thing" node. This is referred to in the "cake 1" procedure above.

<TITLE>Baked Good</TITLE>
<AUTHOR>Colleen Kehoe</AUTHOR>

<DESCRIPTION>
```

```
A baked good is usually found in a bakery. They are things like: bread, cookies, cakes, muffins, etc.
</DESCRIPTION>

<COUNTEREXAMPLE>
<MEDIA SRC="fish.mov" CAPTION="While a fish can be baked, it is not considered to be a baked
good."/>
</COUNTEREXAMPLE>
# Here, we provide a counterexample to a baked good.
# It is in an external media file, but we provide a textual caption as well.

</THING>
#This marks the end of the thing node.

<STATE id="cake didn't rise">
# Now we define a "state" node. This was one of the problem states referred to in the
# "cake 1" procedure.

<TITLE>Cake didn't rise properly</TITLE>
<AUTHOR>Colleen Kehoe</AUTHOR>

<DESCRIPTION>
The cake didn't rise above the edge of the pan. This is usually caused by accidentally leaving out the
baking powder or salt.
</DESCRIPTION>

<LINK type="repair-procedure"> # These are links to repair procedures.
<TARGET id="eat it anyway"/> # These are "procedure" nodes.
<TARGET id="feed to birds"/>
</LINK>

<LINK type="related-to"> # This is a link to a related node (here, a similar problem).
<TARGET id="cake cracked"/> # This is a state.
</LINK>

</STATE>

# Other procedures, things, and states are defined similarly.

</PML> # This marks the end of the PML document.
```