PML: Adding Flexibility to Multimedia Presentations

Ashwin Ram, Richard Catrambone, Mark J. Guzdial, Colleen M. Kehoe, D. Scott McCrickard, and John T. Stasko
Georgia Institute of Technology

In multimedia systems, designers typically link content and presentation. A new markup language, Procedural Markup Language (PML), decouples content and presentation. It lets users specify the knowledge structures, underlying physical media, and relationships between them using cognitive media roles. This approach fosters modular system design and dynamic multimedia systems that can determine appropriate presentations for a given situation by allowing knowledge specification to be done separately from knowledge presentation.

Suppose you're a homeowner faced with a leaky faucet. If you're reasonably comfortable with home repair, you might just plunge in and fix it. If not, you might seek help from a book or perhaps one of the new home-repair multimedia guides for your PC. Will the guide present the information you need at the level you need it? If you're confident in home repair, but don't know faucets, you probably want guidance on key steps, without a lot of detail. If you're a novice, however, you're more likely to need examples, including photos and diagrams, and extensive explanations.

This hypothetical scenario raises two key issues in multimedia system development:. How can information be dynamically presented to meet users' needs and prior knowledge? How should content be encoded so that multimedia developers can present information tuned to the audience and adaptable to different devices.

Typically, designers create a system in which content and presentation are inseparably linked; they choose specific presentations and navigational aids for each piece of content and hard-code it into the system. However, we think content representation should be decoupled from presentation design and navigational structure. This decoupling permits modular system design, and lets designers build dynamic multimedia systems that can determine appropriate presentations in a given situation on the fly.

To this end, we developed Procedural Markup Language (PML), which uses cognitive media roles to flexibly specify the knowledge structures, the underlying physical media, and the relationships between them. The PML description can then be translated into different presentations depending on the context, goals, and user expertise.

Here we describe PML, focusing specifically on procedural task domains, with examples from home plumbing procedures. The highlights of our formalism are:

- Domain information is encoded in knowledge nodes that are connected to each other through knowledge links.
- Information within a particular knowledge node is represented using physical media clusters that contain 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" and "example." A cognitive media role can contain more than one different physical media type. For example, a faucet might be represented using some text and a graphic.
- The information contained in knowledge nodes, knowledge links, physical media clusters, and cognitive media roles forms the raw material of presentation; it determines what users will see and hear, and which navigational connections and devices will be available onscreen. Different presentations can be created from the same underlying representation based on various factors such as the user's domain expertise, previous experience, and goals.

Issues in system design

Content representation is a central issue in multimedia system development, whether on the Web or as a stand-alone system. The content represented is the information presented to users. For example, an educational system to teach 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, you must attend not only to the content, but also how to present it. For example, should information about show-

1070-986X/99/$10.00 © 1999 IEEE

ers be presented graphically? Should information about installing memory chips appear in an itemized list? Should information on gas laws employ animated video clip showing the movement of molecules as a gas is compressed? Typically, content and presentation are coupled: the designer selects a specific presentation for each piece of information—perhaps using several media such as text, pictures, and sound—and hard-codes this into the system so it displays in the same form each time. Designers often choose specific navigational aids as well so that a predetermined hypermedia structure is encoded and available to the user.

However, our design philosophy treats knowledge representation and presentation design as separate activities. In this way, the knowledge engineer or content designer can focus on knowledge representation and structure. The presentation designer can focus on the multimedia presentation of this knowledge, deciding, for example, how to display a diagnostic procedure and at what detail and expertise level, as well as what the display might link to, what navigation aids are needed, and what user-related issues might arise concerning media comprehension.

Our approach has several benefits. First, it's easier for domain experts (who may not be presentation experts) to build the knowledge representation without having to worry about how the information will ultimately be displayed. Second, it lets designers build different presentations for the same information based on user expertise, goals, and other factors.

Finally, our approach lets designers build truly interactive multimedia systems in which the system creates appropriate presentations on-the-fly based on the current interactions and context. Only the knowledge must be specified beforehand. Whether a diagnostic procedure, for example, appears as an itemized list of textual bullets, a graphical flow chart, an animated movie, or some combination thereof can be determined dynamically. When knowledge and presentation are tightly coupled, each presentation 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's important to provide a principled means of coupling the knowledge representation structures with the underlying media, but in a way that provides the flexibility required for interactive and dynamic presentations.

To this end, we organize media according to their *cognitive role* (the function they play in the user's cognitive processes).[1] Consider, for example, a student using an educational multimedia system to learn chemistry, or a homeowner using a home repair Web site to help fix a leaky faucet in a bathroom. The user is unlikely to think, "I'd like to see some text now" or "I could really use a WAV sound file now." The user is more likely to think, "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.

Given this, we organize and couple multimedia content 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 user's cognitive processes.

Researchers have made significant attempts 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 knowledge base 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, such as keeping link information separate from the multimedia document itself.

Our work contributes to existing science in two ways. First, our use of cognitive media roles represents a theory of effective organization for instructional multimedia. Second, we contribute the PML notation, which encodes knowledge so that designers can generate an effective representation.

Knowledge representation

Figure 1 summarizes PML's basic knowledge structure, which includes knowledge nodes, knowledge links, physical media clusters, and cognitive media roles. The information contained in these elements forms the raw material used by a presentation system to determine what users will see and hear, and what navigational connections and devices to make available on the screen.

Figure 1. Knowledge representation in PML. Definition and Example are cognitive media roles and can have more than one associated physical media cluster, such as Audio and Graphic.

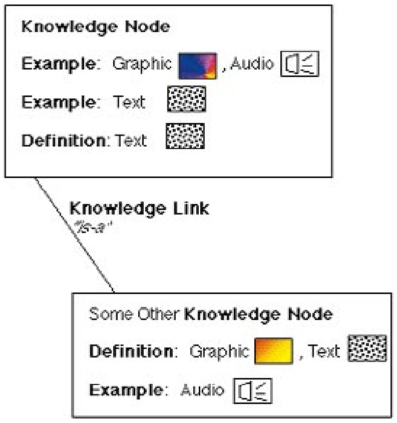


Table 1. Knowledge nodes.

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. A thing's normal state is its usual operational state; other states represent problem conditions that may need repair. Usually, only problem states are represented explicitly; a thing's main representation is assumed to be its normal state.	Pilot light off Faucet leaky Washer worn out Water is brown
Procedure	Represents a sequence of user actions that operate on a thing. Procedural actions 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

Table 2. Knowledge links.

Name	Description	Examples
Is-a	Thing *is-a* Thing Represents the broader category of a thing, or (in 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 (in 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 Represents the substeps of a procedure: steps that represent the procedure in more detail (these steps can 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*.	Replace washer *steps* (Unscrew nut; remove washer; insert new washer; replace nut) Unscrew nut *step-of* Replace washer
Problem-state	Thing *problem-state* State Links things to their possible problem states. A problem state is an abnormal state that r equires 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 Faucet 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	Pilot light off *repair-procedure* Relight pilot light

Knowledge nodes

In PML, a knowledge node represents a concept known to the system. Media clusters contain information about the concept, and knowledge links represent the relationships among concepts. PML's structure resembles the semantic net structures used for reasoning in artificial intelligence systems. However, unlike semantic net systems, PML nodes do not contain slot-filler concept representations. For multimedia presentations, nodes must contain media to present concepts to the user.

In keeping with basic ontological principles, we divided represented entities into things, states, and procedures. Table 1 describes each in more detail. Based on our experience with several different domains, this ontology seems sufficient to capture the distinctions needed in our target domains, which primarily concern procedural knowledge. In general, however, PML's robustness can only be determined after using this formalism in many domains.

Knowledge links

Knowledge nodes can be linked to other knowledge nodes using knowledge links that represent conceptual relationships between nodes. Table 2 describes the links and offers examples of each.

Knowledge links are strongly typed. For example, the `precondition` link always connects states to

(Table 2 continued)

Name	Description	Examples
	installation procedure is also represented as a repair procedure. The reverse link is *repair-procedure-for.*	Relight pilot light *repair-procedure* 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 can be presented to the user as evidence that the procedure was successfully carried out. A procedure can 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 can link a state to another resulting state. 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 the node that represents that action. The reverse link is *requires.*	Pilot light off *precondition* Open water-heater access panel Open water-heater access panel *requires* Pilot light off
Uses	Procedure *uses* Thing Links procedures to tools, instruments, and other required objects. The reverse 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 related or relevant node. 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

Table 3. Cognitive media roles.

Name	Description	Examples
Name/Title	The (typically short) name of the knowledge-node item.	"Pilot light."
Definition/ Description	The knowledge-node concept's definition, or more informally, its description. This is typically 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 knowledge-node concept.	[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, and so on.	[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..."

procedures. You can list multiple links of the same type under nodes in any order, with the exception of **steps**, which must be listed in the order in which they should occur. The system can traverse knowledge links in either direction, although each link has an explicit source and destination endpoint. Table 2 also lists the reverse links, which are static and managed by the system, rather than created by the knowledge designer. As with the knowledge nodes, the link set has proven sufficient in several domains, but we have yet to test it widely.

Physical media clusters

A physical media cluster contains the knowledge-node information that the system can display to the user. Media are stored in separate files, referenced via the **Media** tag, with the exception of text, which—for convenience sake—can be included inline in the PML document. A media cluster can contain more than one media type, such as text, video, and so on. A knowledge node can contain one or more media clusters, organized using cognitive media roles.

Cognitive media roles

Media roles connect knowledge structures and media clusters by indicating the role the cluster plays in the user's problem-solving task. For example, a particular mixed media text-and-pictures description of a faucet might offer an "example" of a single-lever faucet. In this case, the knowledge node for "single-lever faucet" will contain an "example" role that contains a media cluster representing that text-and-pictures description. Table 3 gives descriptions and examples of PML's five cognitive media roles.[1]

Figure 2 shows how the nodes and links fit together, using a piece of the faucet representation from the home repair domain.

PML notation

PML lets authors encode our knowledge representation in a set of files. PML authoring is analogous to that in Hypertext Markup Language (HTML) or other markup languages, but with a crucial difference: The PML author focuses on representing domain information rather than presentation information. In PML, the two types of information are decoupled. For example, if you were to use HTML to create a Web page describing how to install a software release, it might look something like this:

```
  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, which you must state using a particular physical representation (in this case, a numbered list of items). Using PML, you would specify the steps independent of the presentation:

```
  <PROCEDURE ID="install">
  <TITLE>Installing the
```

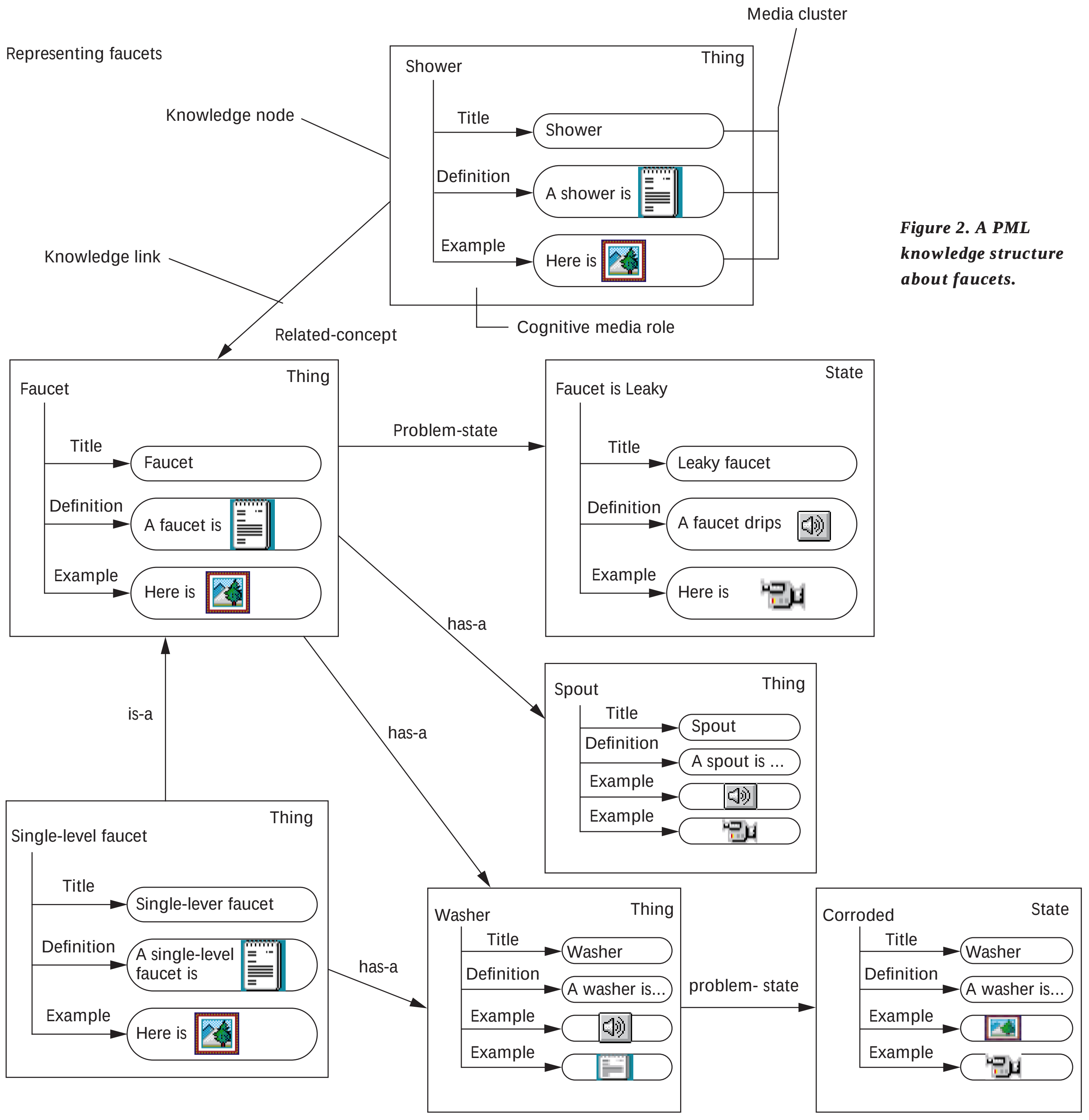


Figure 2. A PML knowledge structure about faucets.

```
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, which PML also represents. 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; the user can make that decision independently and, if desired,

```
<!-- 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>
```

Figure 3. The PML Specification.

dynamically (limited only by the physical media provided). PML also lets the author represent different types of information, such as the procedure's preconditions and outcomes, things that might go wrong and how to recover from them, justifications for the procedures, and so on. Although these can also be hard-coded into the HTML representation, the presentation would be static. Finally, the PML represents procedures and subprocedures hierarchically; the level of detail can be determined dynamically depending on factors such as the user's expertise.

PML is written in Extensible Markup Language (XML),[5] a language for describing other markup languages. A markup language is essentially a set of tags that an author uses to describe parts of a document. 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>`**, and **`<IMG>`**. While these tags are useful for describing basic document structure, they don't describe the content of a document very well. More powerful and most likely domain-specific markup languages are needed for this purpose.

XML is a simplified version of Standard Generalized Markup Language (SGML), an international standard for creating structured documents. XML was developed as a common way to define different markup languages, though it has utility far beyond the World Wide Web. XML can also be used as a platform-independent way to represent knowledge in a machine-readable format.

XML has 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 lets developers build tools that can work with any of the languages specified in XML. Our PML-to-HTML translator uses a PML parser, which is actually generic given that it understands XML and therefore any markup language specified using XML.

The notation used in XML descriptions is fairly standard. Each **`ELEMENT`** statement is a production rule, with the first item to the left of the rule and the second item (in parentheses) to the right. Each 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 of a particular element, such as a tag. It also specifies the attribute's type and whether it's required (**`REQUIRED`**) or optional (**`IMPLIED`**). Figure 3 shows the complete PML specification; Figure 4 shows an annotated PML sample from a common procedural domain: cake baking.

```
<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.
```

Figure 4. An annotated PML sample from the cake-baking domain.

(Continued next page)

Figure 4 continued

```
<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.
```

PML authoring tools

Typing in PML structures can be a tedious and mistake-prone process. We thus developed tkPML, a graphical editing tool for creating the knowledge node/link networks graphically. Figure 5 shows a tkPML session.

The tkPML graph-creation interface lets designers enter node and link information using a point-and-click interface, with forms for entering text. In addition to the obvious benefits, tkPML might also help

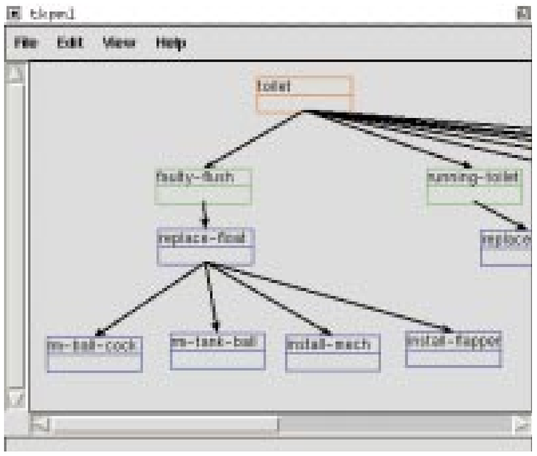
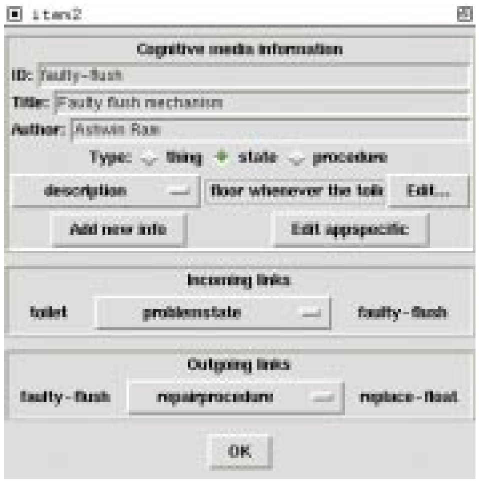


Figure 5. A screenshot of a tkPML session. Designers create nodes by double clicking on the background and move them by dragging the node's top "title" portion. Links are created by dragging from the bottom "link" portion of one node to another. Double clicking on an existing node pops up a node information screen (shown in Figure 6, next page), which lets designers edit node information.


itsm2

Cognitive media information

ID: faulty-flush

Title: Faulty flush mechanism

Author: Ashwin Ram

Type: thing [✓ state] procedure

description | floor whenever the toil | Edit...

Add new info | Edit appspecific

Incoming links

toilet | problemstate | faulty-flush

Outgoing links

faulty-flush | repairprocedure | replace-float

OK

Figure 6. View of a knowledge node in tkPML. The top of the screen shows the default cognitive media roles. These can be edited, or others can be added.

designers better establish and maintain mental models of their PML representations.

In tkPML, designers can create and position nodes and links with simple mouse actions. They can view the layout at various levels of detail, from an overview of large knowledge structures to a detailed view of a few nodes. Also, as Figure 6 shows, each node can be expanded to view and change its knowledge.

The tkPML tool saves a PML description file as a comment in an augmented PML format file, adding a simple node location to the standard PML definition. Thus, designers can edit the saved files by hand or run presentation interpreters on the files without modification. without modifying the files. In addition, tkPML can import files written in PML, even if they weren't created using tkPML. For such files, tkPML generates an initial display using a simple graph-layout algorithm.

TkPML is written in Tcl/Tk, a platform-independent graphical scripting language that can run on Unix, PCs, and Macintoshes, and in Web browsers. Designers can thus freely exchange and edit PML representations.

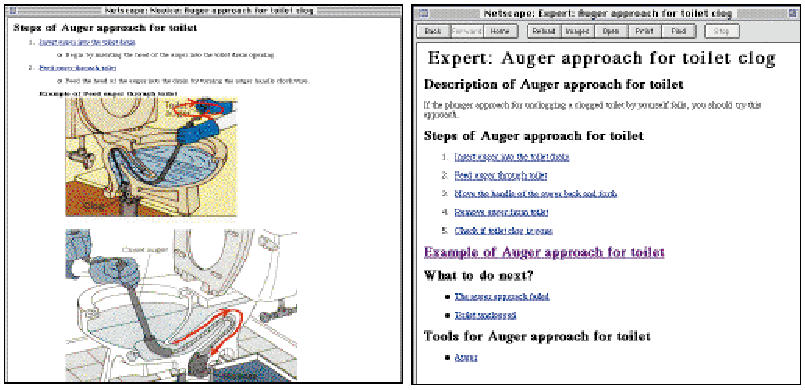
Presentation tools

There are two ways to create a presentation based on a PML document. First, you can develop a PML interpreter-generator that can interpret PML descriptions, retrieve the appropriate media within them, and create a hyperlinked presentation based on users' situation and needs.

For example, if a novice user is simply trying to determine whether to call a plumber to repair a leaky shower, she could initially choose a summary of the time, expertise, and tools needed, rather than a display of all the repair details. If the same user has previously repaired a leaky faucet, she can select a top-level overview of the general leak-repair procedure, using links to substeps that differ. Truly interactive and dynamic multimedia systems will need such capabilities. We designed PML to support the very design principles of instruction and interaction that form the basis of this type of interpreter-generator.

The second and simpler option is to develop a PML interpreter that can construct a few predetermined presentations. For example, you might develop one that always displays substeps in detail, and another that always displays substeps as titles with links to the details. You could then use a simple heuristic to decide which presentation to use.

We chose this second approach for our initial implementation, a PML-to-HTML translator that can create presentations based on simple, predetermined presentation rules. Figure 7 shows two such presentations, which we designed to help users unplug a toilet drain using an auger. Figure


Netscape: Novice: Auger approach for toilet clog

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

Back | Forward | Home | Reload | Images | Open | Print | Find | Stop

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

(a) (b)

Figure 7. Two presentations based on the same PML source. (a) The presentation aimed at the novice offers detailed explantions and examples; (b) The one for an expert is more concise, but with links to further explanations.

7a shows a presentation aimed at a novice; Figure 7b shows one for 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, we provided more introductory information (not shown here), such as what an auger is, and for each step offer explanations and examples, as the figure shows.

With both presentations, we assumed that the target platform was a Web browser on a PC. 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, we would create a different structure. With a handheld, for example, we would not use a long scrolling presentation like that in Figure 7b.

Conclusions

Cognitive media roles have been used successfully 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 as a whole has been used for other tasks and domains. For example, 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] Although not dynamic, these early systems illustrate the generality and value of the knowledge representation and the notational tools.

In our current research, we are developing PML-based systems to investigate cognitive issues in dynamic multimedia system design. More broadly, we are investigating how the goals that students bring to a learning task can affect their learning processes. We will use this knowledge to develop a hypermedia support system that can adjust itself in response to a user's goals.

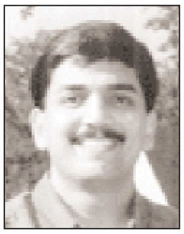
PML lets us examine these and other issues empirically. For example, we would like to systematically study which factors help determine the effectiveness of different presentations for the user. Is it true that a "high-level" presentation of a procedure for a more knowledgeable person is more effective than one that provides all the details? Because PML can help us create alternative presentations, we can begin to empirically explore such issues. MM

Acknowledgments

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.

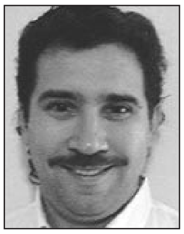
References

1. M. Recker et al., "Cognitive Media Types for Multimedia Information Access," *J. of Educational Multimedia and Hypermedia*, Vol. 4, No. 2/3, 1995, pp. 185-210.
2. M. Maybury, ed., *Intelligent Multimedia Interfaces*, MIT Press, Cambridge, Mass., 1993.
3. S. Feiner and K. McKeown, "Automating the Generation of Coordinated Multimedia Explanation," *Computer*, Vol. 24, No. 10, Oct. 1991, pp. 33-41.
4. I. Tomek, H. Maurer, and M. Nasser, "Optimal Presentation of Links in Large Hypermedia Systems," *Proc. 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-Feb-1998, http://www.w3.org/TR/1998/REC-xml-19980210.html.
6. R. Cover, "XML: Proposed Applications and Industry Initiatives," 1998. http://www.oasis-open.org/cover/xml.html.
7. G. Shippey et al., "Exploring Interface Options in Multimedia Educational Environments," *Proc. Second Int'l Conf. on the Learning Sciences*, Assoc. for Advancement in Computing Education, Univ. of Virginia, Charlottesville, Va., 1996.
8. M. Byrne et al., "The Role of Student Tasks in Accessing Cognitive Media Types," *Proc. Second Int'l Conf. on the Learning Sciences*, Assoc. for Advancement in Computing Education, Univ. of Virginia, Charlottesville, Va., 1996.
9. M. Guzdial, "Technological Support for an Apprenticeship in Object-Oriented Design and Programming," *Proc. OOPSLA 97 Educators Symp.*, ACM Press, New York, 1997.
10. M.T. Realff et al., "Multimedia Support for Learning Advanced Packaging Manufacturing Practices," *Proc. ECTC 98*, IEEE Computer Society Press, Los Alamitos, Calif., 1998.



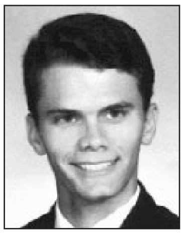
Ashwin Ram is an associate professor of computer science and cognitive science at the Georgia Institute of Technology. His research interests are in intelligent systems, cognitive science, machine learning, and natural language understanding. He is co-editor of *Goal-Driven Learning* published by MIT Press/Bradford Books in 1995, and *Understanding Language Understanding: Computational Models of Reading*, published by MIT Press, 1999. Ram received his BTech in electrical engineering from the Indian Institute of Technology, New Delhi, in 1982, his MS in computer science from the University of Illinois at Urbana-Champaign in 1984, and his PhD in computer science from Yale University in 1989.

Richard Catrambone is an associate professor in the School of Psychology at Georgia Tech. His research interests are in problem solving, analogical reasoning, and human-computer interaction. He is particularly interested in how people learn from examples in order to solve problems in domains such as algebra, probability, and physics. Catrambone received his BA from Grinnell College in 1982 and his PhD in experimental psychology from the University of Michigan in 1988.

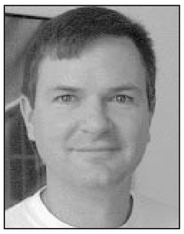


Mark Guzdial is an associate professor in the College of Computing at Georgia Tech. His research is in technological support for project-based learning and computer-supported collaborative learning. He received his PhD in education and computer science from the University of Michigan in 1993.

Colleen Kehoe is a PhD student in the Graphics, Visualization, and Usability Center at Georgia Tech. Her research interests include educational technology, design education, user interfaces for educational software, and Web-related technologies. She received her BS in computer science from Stevens Institute of Technology in Hoboken, N.J. in 1994.



Scott McCrickard is a PhD candidate in the College of Computing and a member of the Graphics, Visualization, and Usability Center at Georgia Tech. His interests include user interface design, information visualization, and multimedia communication methods.



John Stasko is an associate professor in the Graphics, Visualization and Usability Center and the College of Computing at Georgia Tech. His research interests include software and information visualization, human-computer interaction, programming environments, and software agents. He received a BS in mathematics from Bucknell University in 1983 and ScM and PhD degrees in computer science from Brown University in 1985 and 1989, respectively.

Readers may contact Ram at College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, e-mail ashwin@cc.gatech.edu.