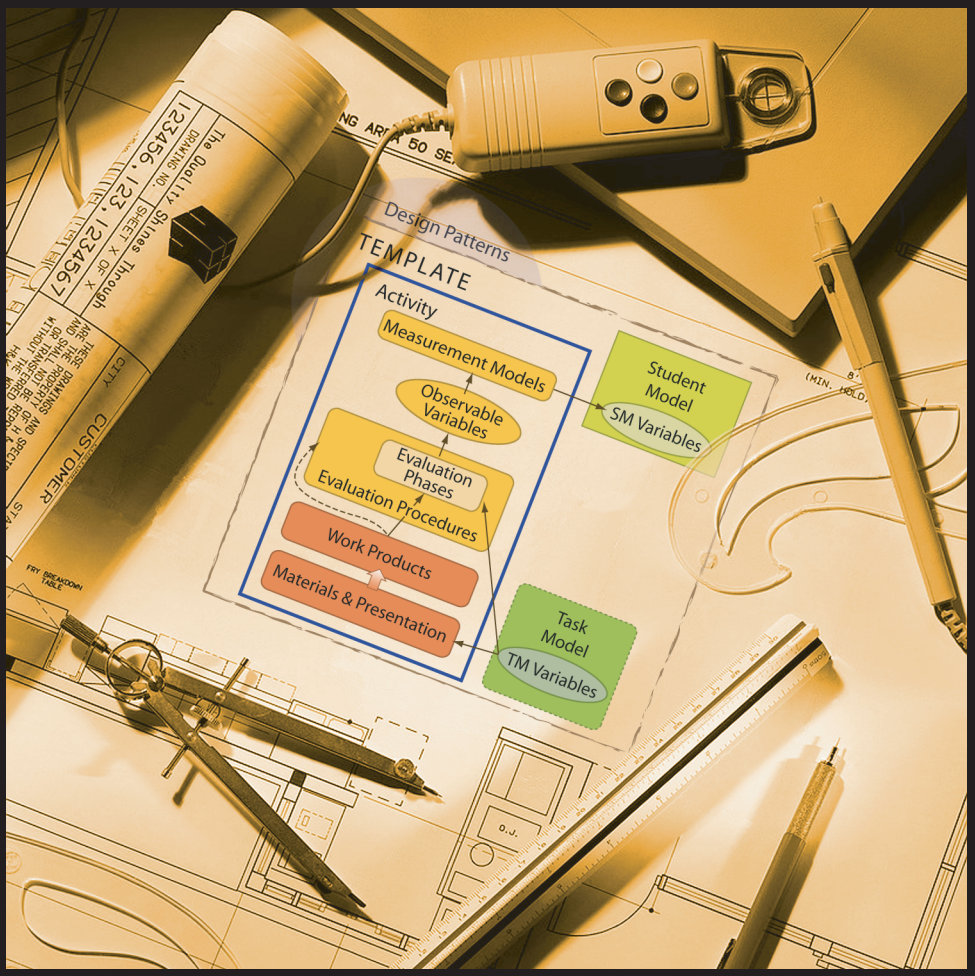



PADI Technical Report 4 | March 2005

# Participatory, Example-Based Data Modeling in PADI

PADI | Principled Assessment Designs for Inquiry

**Lawrence Hamel**, CodeGuild, Inc.

**Patricia Schank**, SRI International

Report Series Published by SRI International

**SRI International**
**Center for Technology in Learning**
**333 Ravenswood Avenue**
**Menlo Park, CA 94025-3493**
**650.859.2000**
**http://padi.sri.com**

**PADI Technical Report Series Editors**
Alexis Mitman Colker, Ph.D. *Project Consultant*
Geneva D. Haertel, Ph.D. *Co-Principal Investigator*
Robert Mislevy, Ph.D. *Co-Principal Investigator*
Klaus Krause. *Technical Writer/Editor*
Lynne Peck Theis. *Documentation Designer*

PRINCIPLED ASSESSMENT DESIGNS FOR INQUIRY
TECHNICAL REPORT 4

# Participatory, Example-Based Data Modeling in PADI

Prepared by:
Lawrence Hamel, CodeGuild, Inc.
Patricia Schank, SRI International

*Acknowledgment*
PADI is supported by the Interagency Educational Research Initiative (IERI) under grant REC-0129331 (PADI Implementation Grant).
*Disclaimer*
Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## ABSTRACT

Domain experts are essential for successful software development, but these experts may not recognize their ideas when these are abstracted into Unified Modeling Language (UML) or ontologies. We describe a Web-based tool for modeling that creates and manipulates a simple data model without representing it in UML, while promoting collaboration and the use of examples to compare and validate the model. The free, open-source tool, "EMo," is a by-product of a team effort to invent and refine a complex data model and library of examples for the Principled Assessment Designs for Inquiry (PADI) project. We discuss alternative tools, such as UML editors, as well as the process that led to EMo.

# 1.0 Introduction

## 1.1 Modeling with Collaboration and Validation

In an effort to standardize digital communication in their fields, many usage communities and trade groups have created Extensible Markup Language (XML; Goldfarb & Prescod, 1998) data models with acronyms like HL7 (Health Level Seven, Inc., 2004), QTI (IMS Global Learning Consortium, Inc., 2000), and FIX (Fix Protocol, Ltd., 2004). The collaborative design of these data models is often the primary activity of such groups, and their published models can have a significant impact. For example, HL7 (hl7.org) is widely used in medical informatics, QTI (imsglobal.org) is popular in learning-management systems, and FIX (fixprotocol.org) is used in securities exchange. Most large commercial systems in their respective domains tout abilities to interchange information with these standards, and their respective companies spend large sums to sponsor and influence the standards bodies.

Within workgroups of domain experts, such as those mentioned above, the contributions of individuals are mediated by tools and representations, and perhaps by technical personnel who run the tools and edit the representations. How do these nontechnical domain experts contribute to modeling and validate that their domain is represented correctly by a candidate model? Where do concrete examples fit into collaboration? New techniques are needed to increase participation of domain experts in data modeling to support creative, interdisciplinary, and collaborative exploration to promote better designs (Sullivan, 2004). Domain experts must be able to validate that their domains are represented correctly by candidate designs. In line with these goals, participatory design—in which stakeholders take a proactive, central role in the design team, working with engineers on a design—can often lead to more usable designs and shortened development and test cycles (Schuler & Namioka, 1993). This paper describes one approach to supporting collaborative data modeling and offers a brief comparison with other popular approaches to data modeling (Sanders, 1995).

The effort to define a data model collaboratively raises issues similar to those that are already well documented in the fields of knowledge engineering and knowledge management: the difficulties of knowledge extraction and the capture of social context. For example, in artificial intelligence research, the development of expert systems relies on extracting knowledge from experts and representing that knowledge in the system. In practice, it is quite difficult and time-consuming for experts to articulate their (often tacit) knowledge and skills, removed from the context of an activity (Dreyfus, 1993). Designers of knowledge management systems experience similar issues when they try to codify employees' situated knowledge within a company knowledge base. After employees leave a company, their knowledge often cannot be recreated because context and practices were not captured (Brown & Duguid, 2000). Researchers in situated cognition (e.g., Brown & Duguid, 2000; Suchman, 1987) argue that context and practice are both critical to understanding knowledge and extremely difficult to represent in knowledge systems.

## *1.2 Background*

The work described in this paper grew out of a National Science Foundation (Interagency Education Research Initiative) project, Principled Assessment Designs for Inquiry (PADI), that attempts to provide a practical, theory-based approach to developing quality assessments of science inquiry and to model the psychometrics of the assessment of scientific inquiry skills (Mislevy et al., 2003). The PADI team includes nearly 30 members from four educational organizations: SRI International, University of Maryland, University of California at Berkeley, and University of Michigan. The vast majority of the team members are domain experts in science education, educational research, assessment, and instruction. When the project began, only a handful of team members were familiar with data modeling or software development.

The main goal of the PADI project is to develop a rigorous design framework for assessing middle school student inquiry skills in science, which are highlighted in various education standards but difficult to assess. A primary deliverable of this design framework is a robust data model of the domain of assessment. Our main goal is not to develop a specific application (like a tool for teachers to create assessments) but to build a framework for creating assessments, including a flexible data model that could serve in many such applications. To inform our efforts, we experimented with sample assessments and assessment tools (like grade books) that used the domain model.

As the project team teleconferenced to discuss models of student understanding, they needed a way to manipulate and compare multiple models and examples in a distributed, shared manner. When manipulating their models, they wished to see the effect on their sample data immediately. The domain experts were not familiar with Unified Modeling Language (UML; Arlow & Neustadt, 2001), the de facto standard for data modeling, nor with ontology construction more generally (Gruber, 1995). The team members charged with software development sought to augment a modeling process with an example-rich approach to modeling. Additionally, the project required the collection of a library of samples, so the authors sought to leverage the creation of exemplars (for validation and discussion) as an opportunity to add to the final library. A Web application, dubbed "EMo" (for Example-based Modeling), was designed to expose a "virtual" data layer, a layer of abstraction between the model displayed and the real database structure (Schank & Hamel, 2004). Domain experts may use this virtual layer to create and modify models and interrelations among models. EMo provides a way for them to browse and enter examples in order to validate the design under discussion.

In this paper, we first review two popular domain modeling approaches—UML and ontological engineering techniques—and explain why they are not ideal for our purposes. Then, we introduce the alternative example-based approach that was developed by our team. Finally, we describe the features of the EMo system that we implemented to support our data modeling needs.

## 2.0 Limitations of Existing Domain Modeling Techniques

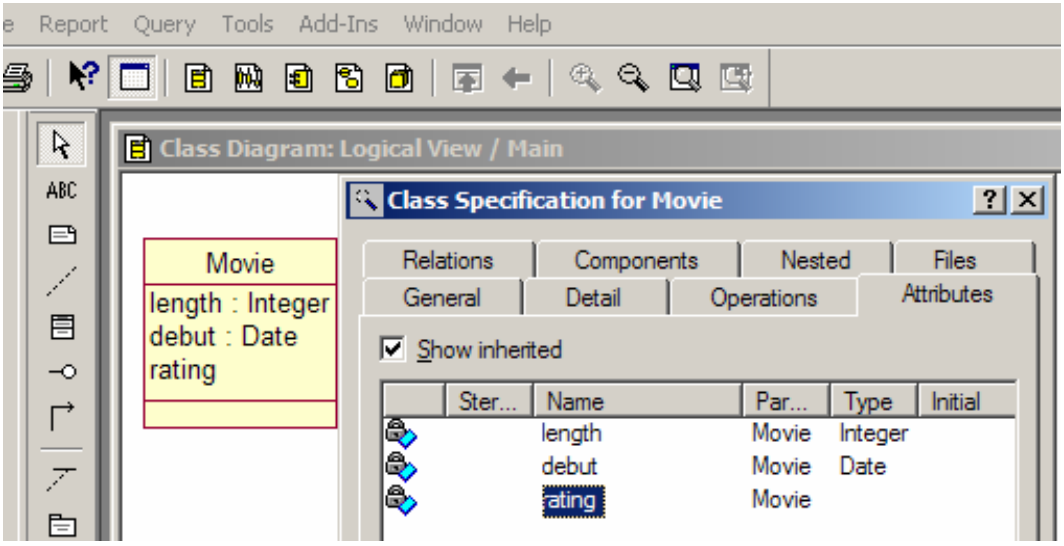
### 2.1 Unified Modeling Language (UML)

One of the earliest methods used for data modeling is Entity Relationship (ER) modeling, which graphically presents the elements of a relational database through sets of shapes and lines (Chen, 1976). The Unified Modeling Language (UML), which shares many surface features with ER, is the de facto standard for object-oriented data modeling (Arlow & Neustadt, 2001). However, UML features take considerable time to learn, and UML is recognized even by its proponents as being complex and difficult to use for communication with nontechnical members of a project (Tilley & Huang, 2003). Commercial tools like Rational Rose (IBM Corporation, 2004) allow visual modeling in UML and similar graphical formats. Models are created with a visual editor, and in Rose, collaboration is supported to the extent that one person can edit while displaying the screen to others, who critique. The editor can also publish pictures of the UML diagrams to the Web. For a teleconference, some kind of screen-sharing technology like NetMeeting (Microsoft Corporation, 2004) might be used to share the application. Another commercial tool, Poseidon for UML (Gentleware, 2004), which is based on the popular open-source ArgoUML project (Tigris.org, 2004), recently began offering an Enterprise Edition that allows the sharing of models in a client-server configuration as long as each concurrent user has a license.

While the diagrams and syntax of UML can express very complex models, UML tools typically do not offer much support for entering sample data to validate the models. Most UML tools are intended for an audience of software engineers who will translate the UML diagrams into objects in Java or other programming languages. But generating objects in Java is considerably different from using sample data to populate and validate a data model. UML does offer the ability to express use-cases and deployment diagrams pictorially, but these use-cases typically explain large-grain interactions between the user and the system.

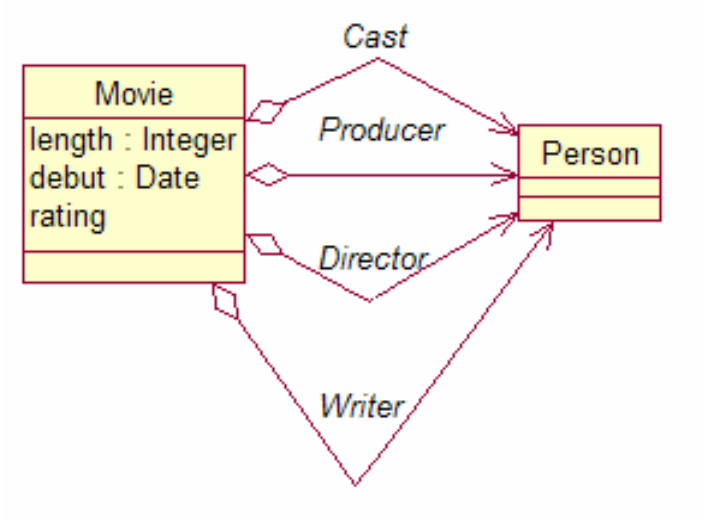
To demonstrate the use of a representative UML tool, consider the following modeling exercise: a “Movie” markup language that might help in interchanging information about movies in XML format. Figure 1 shows the creation of a Movie model in Rational Rose Enterprise, v2002.05.

**Figure 1. Editing a Sample Movie Object Using Rational Rose**




This Movie model will start with only a few attributes: Producer, Director, Writer, and Cast, as well as length, debut date, and rating, as shown in Figure 2. Potentially, roles like Producer and Director may be filled by multiple people, so an appropriate design would use an aggregation relationship; for example, the Movie has a collection of Person instances that constitute the Producer attribute. These aggregation relationships also imply sharing of the Person if, for example, a Director directs two separate Movies. To help distinguish aggregation attributes from others, we will capitalize shared attributes like Cast, leaving other attributes like length in lower-case.

**Figure 2. A Simplified Movie Object in UML**




To share the UML diagram in Rose, the team member who is editing can export the diagram as an image file and transfer it to a Web server. This scenario, however, does not permit shared control of the model or the invention of competing models by others. There is no clear support for creating many instances of the various models within the tool. Rose

customers are expected to create Java instances or SQL schemas and are expected to be knowledgeable about software engineering practices.

## 2.2 Ontological Engineering

In the field of artificial intelligence, ontological engineering tools have been created to help developers and domain experts build effective knowledge-based systems (Domingue et al., 2001; Domingue, 1998; Farquhar, Fikes, Pratt, & Rice, 1995; Gennari et al., 2003; Swartout, Patil, Knight, & Russ, 1996). Some ontological tools attempt to support communities of nontechnical domain experts (Domingue et al., 2001; Gennari et al., 2003), but few emphasize the entry of examples to validate the design. An exception is Protégé (Gennari et al., 2003), which supports complex models and entry of examples ("instances"). However, in Protégé, these sample records, or instances, are reduced to the role of design annotations, with functionality for manipulating and sharing examples buried among many other features. Protégé also recently added a feature for sharing models for collaboration. Because it is a desktop (rather than Web-based) application, however, users must perform additional configuration of Protégé to enable shared modeling. Figures 3 and 4 show screens from Protégé, displaying the same Movie model as shown previously with Rose.

**Figure 3. Editing Movie Attributes Using Protégé**

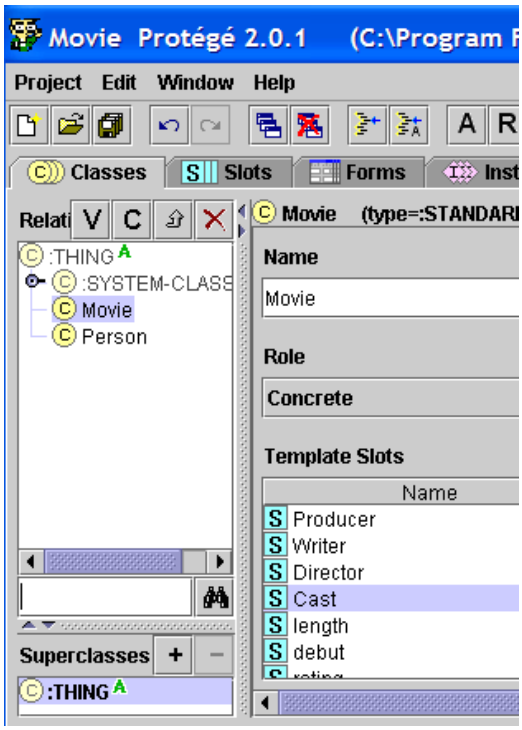
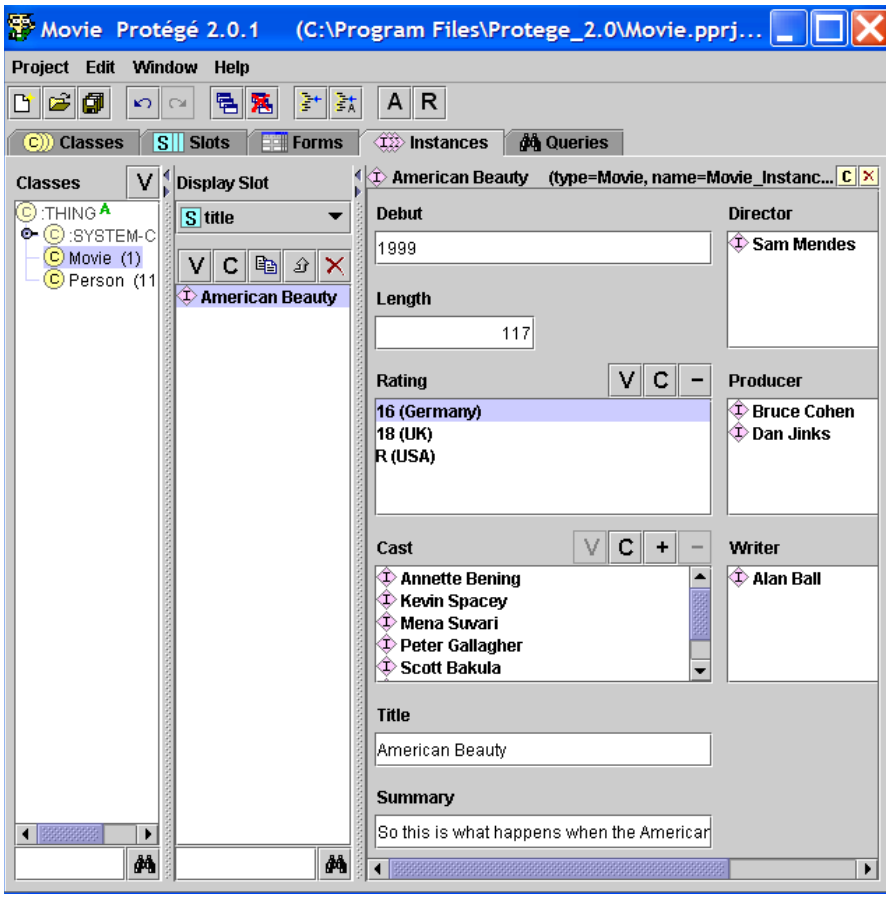


**Figure 4. Entering a Sample Movie Instance Using Protégé**




Protégé allows sample records (instances) to be entered for a given model, as shown in Figure 4. However, the process presents some barriers to nontechnical editors. For example, they are presented with many options and unfamiliar terms in the interface; in the default layout for entering an instance, Protégé may render fields in an unexpected order, and changing that default layout involves programming; and sharing the results with peers involves special configuration.

## *2.3 The Point of View of a Nontechnical Expert*

Now imagine that a teleconference on the Movie model is attended by luminary film directors, each an acknowledged expert in his or her domain. We assume that they will not easily understand the UML diagrams from Rose, because UML features take considerable time to learn (Tilley & Huang, 2003). Although the Protégé entry screens seem easier to comprehend, we expect that they still would be unusual and off-putting for our group of Scorseses and Spielbergs. How will the directors show their sample records to their colleagues in other cities? How will they propose an alternative model to a peer? These are the questions we needed to address in our research group, with a simpler, more example-rich, and sharable tool for modeling, albeit one with less capability to formulate complex models. We attempted to simplify the presentation of models, relying on the common knowledge of Web forms to assist recognition and participation by nontechnical domain experts.

## 3.0 Example-Based Modeling (EMo)

EMo is a Web application that enables data model creation and manipulation with just a Web browser. Team members may be given various permissions that dictate their ability to edit models, enter examples, and view examples. Model editors can create new models, or add attributes to existing models, by typing in a name and description for the new model or attribute. Other team members may have permission only to enter sample records — i.e., populate instances of models. Others may only be able to view sample records. All sample records for a given model gain and lose attribute fields dynamically as the model editors manipulate the models.

In the following, we describe how EMo is best characterized as an instance of example-based modeling and summarize how EMo was developed and the nature of its use by the PADI project team. Then we revisit the Movie example using EMo and highlight the features of the system.

### 3.1 The Role of Examples in Domain Modeling

In contrast to existing systems, EMo is specially oriented around examples as the primary device for communication and negotiation. The research literature indicates that such contextualization can be advantageous. Research on learning has shown that people strongly prefer to learn from concrete examples over abstractions when given a choice (Recker & Pirolli, 1995; LeFevre & Dixon, 1986); and, more importantly, focusing on examples, rather than abstract representations, can both enhance and accelerate comprehension and learning (Chi et al., 1989; Pirolli & Anderson, 1985; Zhu & Simon, 1987) because of issues related to working-memory capacity and motivation. In software design, the use of examples can make design activities more accessible to a great variety of contributors; help designers recognize, capture, and reuse generalizations; and ultimately enhance the effectiveness of the products of design (Carroll, 2000; Carroll & Rosson, 1992). Examples take center stage in scenario-based design, which moves away from the more technology-driven design perspective of abstract descriptions that focus on generic types to a more work-driven perspective of concrete, colloquial examples that focus on a particular user's view and activities that need to be accomplished. Such scenarios provide a common ground for communicating and conveying users' needs and mental models to the developers, helping to create system models that are meaningful and accessible to both groups (Carroll, 2000; Carroll & Rosson, 1992). Similarly, we argue that in domain modeling, examples not only validate a candidate model, they also support the creative interplay among domain and technical experts, improving communication and understanding of requirements and stakeholders' practice and ultimately improving the usefulness and accuracy of the final model.

### 3.2 Design Process

The primary features of EMo were designed and developed over the period of a year in a series of weekly teleconferences that involved the entire PADI team. One of our first tasks was to explore tools and methods to help our team of domain experts articulate a conceptual framework for assessment design, delivery, and scoring. We followed a participatory, iterative design process, integrating top-down approaches that emphasize

concepts and planning with bottom-up approaches that emphasize data and coding. For example, in the top-down approach, content experts drafted user scenarios and software engineers created UML diagrams of the salient concepts like "Student Model," "Design Pattern," and "Measurement Model," showing their interrelations. In the bottom-up approach, software engineers developed prototypes and content experts attempted to enter assessment examples into these prototypes.

During weekly teleconferences in the first 3 months of the project, we focused on defining use-cases to clarify the bounds and scope of the work. During this phase, every team member was asked to submit at least one use-case for group review. Team members frequently revisited and clarified misunderstandings regarding the goals and target users of the system. Many had difficulty understanding what a data model was; their primary experience was with concrete applications, like test delivery systems, which assume some data model. For example, early use-cases submitted by team members often involved teachers and students interacting with the system to develop, administer, and score assessments. After a few iterations and patient guidance by the principal investigators (especially as software engineers grappled with the complexities of assessing inquiry skills), our final set of about a dozen use-cases focused mainly on highly skilled, psychometrically astute assessment developers interacting with a design system to create templates for assessments.

Armed with the knowledge about the target users and requirements from the use-cases, we proceeded to design the data model. In this phase, which lasted approximately 3 months, we began using UML notation to explore possibilities for the data model. We used a typical knowledge extraction method in our weekly meetings, in which a senior engineer presented UML diagrams and solicited critiques from the domain experts. Group members had difficulty understanding the diagrams, as evidenced by repeated discussion of what the various arrows and shapes (circles, boxes, diamonds) meant. The nontechnical domain experts did not feel comfortable with the notation, and they certainly could not have modified the diagrams themselves (they had neither the Rational Rose software nor the familiarity with the notation to make changes). Hence, in this phase, the engineer translated domain expertise into a rough data model.

The PADI team members needed an easier way to manipulate and compare multiple models and examples in a distributed, shared manner and to see the effect on their samples immediately when the model was manipulated. The principal investigators of PADI wanted a design system that would support creative interplay between domain and technical experts in the development of blueprints for complex assessments. They cited early criticisms of technology-supported templates that surfaced in the assessment community as an additional motivation: working on a template on a computer was a largely solitary activity, and it deterred the exchange required between domain experts and assessment experts when tasks were being designed (U.S. Congress, Office of Technology Assessment, 1992). Such solitary activity was reported to reduce the quality and richness of tasks that were being developed.

The team members charged with software development sought to avoid the pitfalls of previous template systems by promoting the creation of samples for validation and

discussion. This approach was also attractive in that it would yield examples for the eventual PADI library. Accordingly, once a high-level (although very limited) model of the domain had been developed in UML, the technical team began implementation of a Web-based software prototype that would allow any team member to populate the candidate data model with examples. The interface design was informed by use-case sketches that depicted a proposed flow of pages a person would see when doing the task described in the use-case. A Web-based solution was implemented to leverage team members' familiarity with Web forms, navigating Web pages, and refreshing a Web page to view new information in the shared repository.

By the 8th month of the project, the prototype was available to team members, and they began entering examples and critiquing the model to identify missing elements and needed revisions.

### *3.3 Use of EMo*

During weekly teleconferences, it was natural for the team to discuss sample records rather than focus on the abstract model per se. When a new attribute or relation was required in a candidate data model, a programmer would make the change in the Web application for viewing the following week. These changes grew frequent enough to merit the creation of a metalevel of functionality to expose editing of the model itself, and what we call EMo began to take shape. Using EMo editing features, changes to the model could happen during a teleconference, and all members would then see their samples change to reflect a new design (e.g., a new, empty field showing up in their sample records). The system also allows the creation of competing models, along with their own sample records.

Using the prototype over 18 months, the PADI team worked to extend and refine the data model to encompass 15 core models, with numerous attributes and relations among these models, as well as sample instantiations of their use. As of September 2004, 20 team members had entered 871 examples into the system.[1] In contrast, with the UML editing tool, a single team member did all the composing. The ability of the evolving data model to accommodate numerous examples created by a large and varied group of nontechnical domain experts suggests that this example-based, participatory process has led to a fairly robust, widely understood model. It also helps us achieve a secondary goal of providing a library of working exemplars for the PADI conceptual framework.

Halfway through the 18 months, the term "prototype" was consciously dropped, and team members began referring to the tool as the "PADI Design System," which includes the model-editing pages we describe herein as EMo, as well as additional features that are specific to the assessment domain.

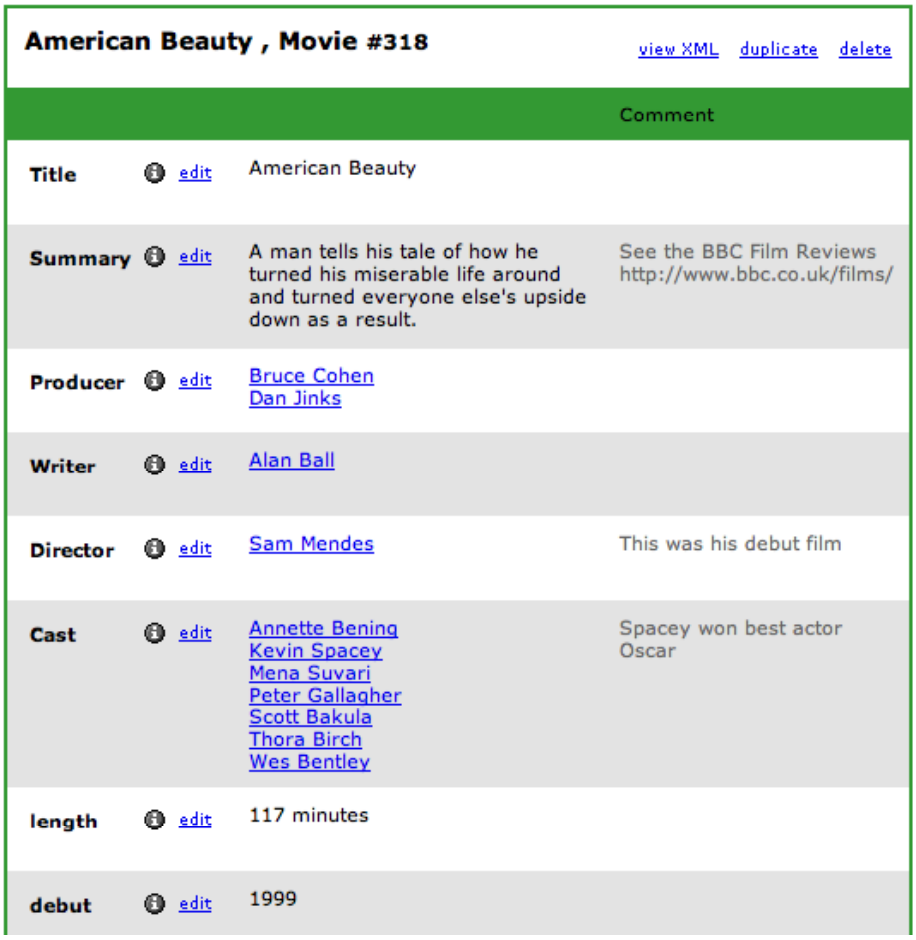
[1] Across model types, the examples are fairly uniformly distributed (median=51, mean=58, min=12, max=125). Across contributors, the distribution is more bimodal, with one outlier contributing 207 examples (median=22, mean=43, min=1, max=207).

## 4.0 EMo Features

Consider the previous Movie modeling example in EMo. Figure 5 shows the editing screen to create a Movie model with several attributes. From this screen, team members can reorder, insert, and delete attributes for a model. A team member could create an alternative model for Movie with different attributes and thereby design on a parallel track. Figure 6 shows a populated instance—a sample record in which data have been entered for a particular movie.

**Figure 5. An EMo Page Allows Manipulation of the Attributes of a Movie Model**

**Figure 6. A Movie Instance in EMo Provides Fields for the Input of Sample Data**




Now imagine our hypothetical teleconference with luminary film directors. When they see Movie instances like that in Figure 6, we expect them to comprehend the data model more quickly than they would by looking at a UML diagram, given the greater difficulty of comprehending abstract UML representations compared with concrete examples. We expect them to tell us that the Director field should be the first field in the display and to ask, "Where, by the way, are the fields for gaffer, grip, and Foley editor?"

## *4.1 Collaboration through Shared Editing and Viewing*

As a Web application with server-side data persistence, EMo offers the potential to share all its contents (given sufficient permissions), including editing rights for the current model and the ability to create a competing model. As model edits are made and stored in the database, all subsequent views reflect the updates, providing immediate feedback to participants.

## *4.2 Shared-Relationship Attributes versus Owned Attributes*

When analyzing a domain in software engineering, one of the first challenges is to identify the primary objects of a domain and the relations between them, as opposed to identifying minor attributes of objects. For example, in analyzing movies, we have an

intuition that producers, location, and script are probably of primary importance, worthy of modeling in their own right and related to the Movie model, while length and sound-processing technology should probably be just minor attributes of the Movie model (of course, this judgment depends on one's perspective, which is why domain experts must negotiate among alternatives to reach consensus).

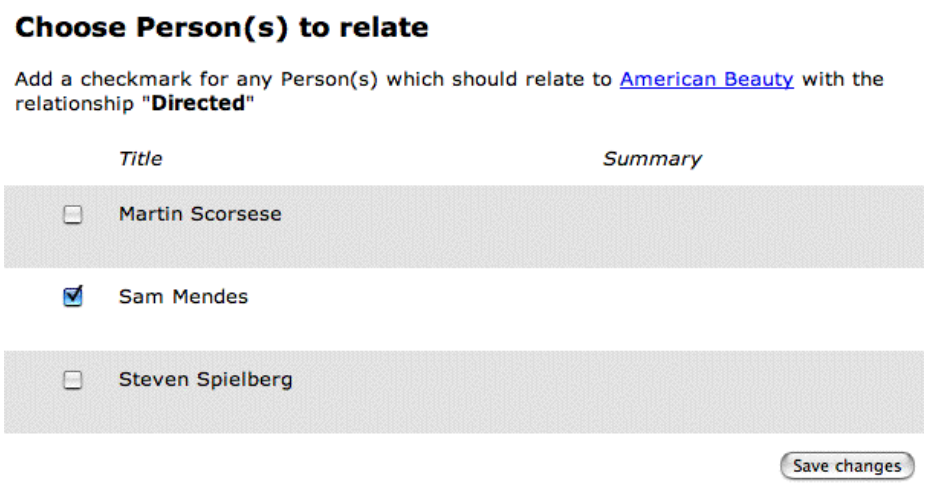
In an effort to make this distinction clearer, we introduced the concept of "shared" versus "owned" attributes. To compare shared versus owned, consider that the same actor can act in both movie A and movie B, but the actor is an instance of the Person model that is related to each movie; Actor is a shared attribute of the Movie model. In contrast, consider that movie A and B can both last 90 minutes, but the length of the movie is not shared, largely because we do not perceive benefit from having "90 minutes" as an instance of some overdone Length model. The length of a movie is simply an owned attribute. Shared attributes are really relationships between instances, and owned attributes are values that belong to one particular instance.

To explain the shared concept to our domain experts, a library of sharable instances was emphasized, wherein users can choose from a list of existing instances to make a relation to the current instance under construction. The modeling of shared relations includes a name for the relation, like "an example of." We prepackaged and emphasized two common relationships, "has-a" (aggregation) and "is-a" (inheritance), although we used the terms "I am a part of" and "I am a kind of" for the labels on these relations, respectively. Relation definitions also specify cardinality, like "only one relation allowed" or "multiple relations allowed." We tended to use the term "shared relations" for shared attributes to save the word "attribute" for "owned attribute."

To explain owned attributes, the concept of identity was emphasized. For example, some attributes of a new instance should not be shared, such as when the attribute helps define the nature of the instance, like the title of the instance. In a sense, owned attributes are defined as whatever should not be shared, from the point of view of the domain expert. Owned attributes are simply edited and viewed as values within an instance.

EMo represents the two types of attributes, shared and owned, with a visual differentiation in both creating and viewing the attribute. In a sample record, shared relationships are created by relating two instances via "checkbox" associations, where the possible candidates are constrained by the model. A list of possible candidates is supplied by the system, as shown in Figure 7. In this example, candidate (Person) instances for the role of Director are presented. The relationships thereby created are viewed as hyperlinks, enabling navigation to view the related instance. Owned attributes are simply edited and viewed as values within the containing instance. In Figure 6, compare the hyperlinks listed for the shared attribute Cast (links to Person instances) with the simple value "117 minutes" listed for owned attribute length (duration of this particular movie).

**Figure 7. Relations Between Instances Are Specified by Checkbox Selections**




When models are designed, how are attributes determined to be shared or owned? This is part of the domain discussion; it depends on how experts view the domain and how they plan to use the data. Taking up the Movie example again, consider the Director of a Movie instance. What should happen if we change the Director's biography, updating some contact information? Do we want this change to stay locally within just one movie? No, that change should be shared, so the information should reside within the Director instance so that all movies sharing the Director instance will also share this updated information.

In contrast, consider the length of a movie. Although a particular length of, say, 90 minutes may hold true for several movies, there seems to be no clear benefit to sharing a library of movie-length instances. If we add 10 minutes to a particular movie instance, we do not want to increase the length of all other movies that also happen to have a length of 90 minutes. (If you can imagine a scenario where sharing movie length is beneficial, by all means, adjust your model to share movie lengths!)

Whether an attribute is shared or owned can be debated and may even change. Fostering well-focused debate is one of the goals of getting domain experts to compare and contrast modeling decisions, even to the point of creating competing models and examples to make their points.

## *4.3 Exporting XML of the Models and Examples*

Although both the models and instances of those models are represented in HTML for manipulation, EMo can export representations in XML. XML tag names are taken from the internal names of the models. For example, if the internal name of the Movie model is "MOVIE" and the internal name of the length attribute is "LENGTH," a segment of XML for a movie might look like Figure 8.

**Figure 8. A Fragment of XML Export for a Movie Instance**

```
<MOVIE NODE_TITLE="American Beauty">
   <LENGTH PART_LABEL="length"
           ATTRIBUTE_VAL="117"/>
      ...
   <RELATED PART_LABEL="Director" ... >
      <PERSON NODE_TITLE="Sam Mendes" ... />
   </RELATED>
      ...
</MOVIE>
```

In the example in Figure 8, the tag <RELATED> indicates a shared attribute. To facilitate documentation and distribution of models, annotated XML of model structures—a kind of glossary of models and their attributes with textual descriptions—is available. Import/export of standards-based XML is also planned for future versions.

### *4.4 Permissions Can Be Specified per Instance (per Row)*

Collaboration groups can be given fine-grained permissions that control reading and writing on a row-by-row basis, where a row in the database corresponds to a model or a model instance in EMo. Furthermore, a distinction is made between those users who have permission to edit examples and those who have permission to edit the model itself.[2] Row-based permissions are supported by the application framework, Expresso (Jcorporate Ltd., 2004), that underlies EMo. Figure 9 shows a code sample of an explicit permissions test for whether or not to show a "delete" link when displaying a particular instance.

**Figure 9. A Test of Privileges**

```
if (node.canRequesterWrite()) {
  // Add a "delete" link
  Transition deleteTransition = new Transition(PROMPT_DELETE_NODE, this);
  deleteTransition.addParam(Node.NODE_ID, node.getField(Node.NODE_ID));
  rowBlock.add(deleteTransition);
}
```

[2] At present, EMo does not have a mechanism for version control. In other words, as with many shared systems, the last person who clicks "Save" has that version saved, replacing whatever content was there previously. In practice, this means that during a conference call, one person is typically selected to be the editor for a given entry at one time.

In Figure 9, "node" is a model instance in EMo, and "Transition" is the Expresso expression of a hyperlink. In this sample, a "delete" hyperlink is conditionally added to the current output if the current user has write permissions. Output blocks like "rowBlock" are created within the controller tier of a three-tier system and subsequently sent to a rendering layer for conversion into HTML or XML.

### *4.5 Menus Can Constrain Attribute Values*

Models may contain attributes that have constrained values (as opposed to free-form text entry), and these constrained values will be presented in a menu. Users with model-editing privileges can dynamically alter the constraints (the menu items) by editing the model. In response, the menus available to model instances will change. For example, if a model editor adds "NC-17" to the possible choices for a Movie's rating attribute, then all screens for editing Movie instances will show this new menu item.

### *4.6 Special Handling Is Possible via Extensions*

Attributes that require custom rendering or special handling for viewing and/or editing can implement a special Java interface (IPartHandler) for that purpose. This feature provides for the creation of arbitrary view/edit screens, custom built for a given attribute. For example, a particular type of attribute might be best represented as a two-dimensional "spreadsheet" of values. In the development for psychometrics, special handlers for spreadsheets and the summarization of nested instances were introduced. Figure 10 shows a code sample for the conditional execution of special handling.

**Figure 10. A Code Sample of a Test to Support Custom Handling of a Model Attribute**

```
if (part.hasCustomHandler()) {

   IPartHandler handler = part.getCustomHandler();

   Transition trans = handler.getEditTransition(request);

   trans.transition(request, response);

   return;

}
```

# 5.0 EMo Implementation

## 5.1 Three-Tier Application within the Expresso Framework

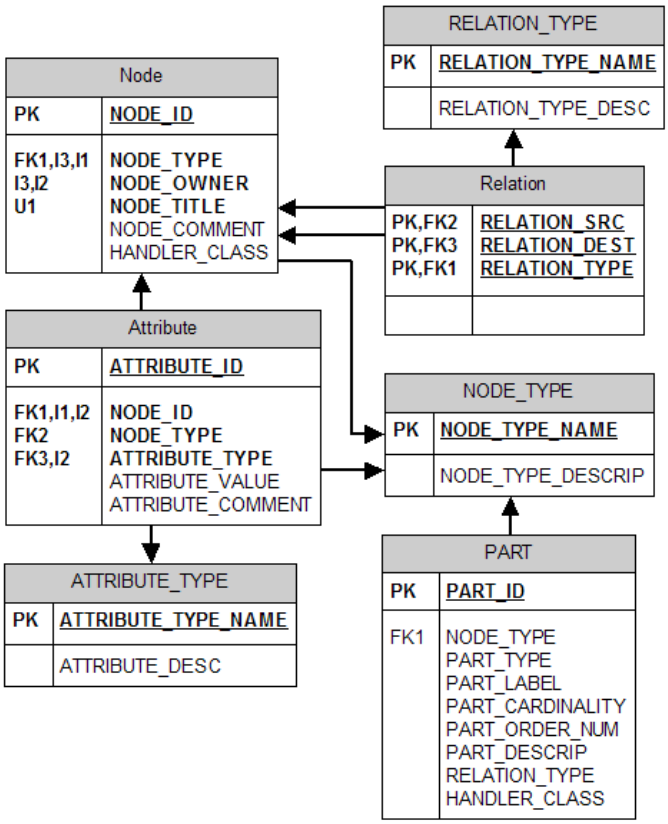
EMo is a three-tier application based on the free, open-source Expresso framework, which in turn is based on Apache Struts (http://jakarta.apache.org) and uses Java Server Pages (Sun Microsystems, 2004) technology for the rendering layer. Expresso offers a model-view-controller (MVC) system, in which the universe of supported Web requests is mapped into a finite-state machine. Expresso includes an object-to-relational layer that permits developers to write persistence code in Java only, without explicit SQL. In practice, this means that our development commonly takes place on desktop computers running Windows or Mac OS X, using the Hypersonic SQL database (Hypersonic SQL Group, 2004), followed by seamless deployment on Solaris servers using the MySQL database (MySQL, Inc., 2004).

## 5.2 Node-Attribute-Relation

Editing a model in EMo actually changes a virtual data layer, a layer of abstraction between the model displayed and the real database structure. A simple database schema underlies the representation on the screen. The database contains nodes, attributes, and relations.

The EMo database schema is shown in Figure 11. The authors acknowledge the irony of showing a UML-like graphic to represent database information within an article about hiding UML. In Figure 11, the tables with uppercase titles pertain to modeling. When creating a new model like Movie, users of EMo actually are creating an entry in the NODE_TYPE table. When adding attribute fields to a new model like Movie, users actually are adding entries in the PART table, like “Director” and “length,” with a foreign relation back to the proper Movie entry in the NODE_TYPE table. A PART table entry can represent either an owned attribute or a shared relationship. Tables that have titles with only initial capitals (Node, Attribute, Relation) pertain to instances of the model. Model instances are stored in the Node table, owned attributes are stored in the Attribute table, and shared relationships are stored in the Relation table.

**Figure 11. A Database Schema for Nodes, Attributes, and Relations**




Every kind of model that is defined by the model editor is defined in Java as a NodeType object with a distinct entry in the field "NODE_TYPE_NAME." For example, the Movie model is a NodeType with the field "NODE_TYPE_NAME" set to "MOVIE." Sample records (referred to as "instances" elsewhere in this paper) are persisted in the Node table and represented in Java with the Node object. Each owned attribute is similarly implemented by an Attribute object in Java with a distinct entry in the field "ATTRIBUTE_TYPE." For example, the length attribute of the Movie model is represented in Java by an Attribute object with the "ATTRIBUTE_TYPE" field set to "LENGTH." Each Attribute object in Java also has a reference ID (a foreign key in database terminology) back to the owning Node.

Relations between NodeTypes are defined by RelationType objects, which can be manipulated by the model editor. In the Movie example, a shared relation like Directed is represented in the database with an entry in the RELATION_TYPE table and in Java with a RelationType object. An actual instance of a Relation is persisted in the Relation table, and represented in Java by a Relation object, connecting one Node Java object with another Node Java object.

One benefit of a node-attribute-relation foundation is that all model instances can be handled by the same functions. The code for manipulating a Movie instance is the same as that for editing a Person instance. For example, the code snippet in Figure 12 collects all the information necessary for displaying all the attributes (parts) for any node.

**Figure 12. A Code Fragment Demonstrating Uniformity in Handling All Types of Nodes**

```
Part[] parts = PartsFactory.getParts(type); // "type" specifies kind of node

Block partList = new Block("allparts");

for (int i = 0; parts != null && i < parts.length; i++) {

   partList.add(getBlock(nodeId, type, request, parts[i], titleStr,

canEdit));

 }
```

One potential disadvantage of this design has to do with the number of records per table. Since all types of objects share the same tables, the tables could become large and unwieldy. However, typical databases perform well with tens and even hundreds of thousands of records, so this design is generally sufficient for research purposes and data model validation, and it meets the needs of the PADI project well.

## 6.0 Future Directions

Many additional features are planned for EMo, including improved filtering, searching, and sorting of instances, improved visualization of the relationships between instances, improved tools to annotate and rank instances, improved data-typing of fields, and import/export of XML schemas in standard formats.

In future research, we would like to observe more domain experts working with EMo, especially nontechnical experts in a team setting. We would like to collect data on usage patterns and observe modeling collaboration in several environments. Because the software is available freely under an open-source license, we hope that others will adopt and adapt it for their own uses and share their feedback with us. Thanks to generous support from the principal investigators of the PADI project, EMo is available at http://sourceforge.net/projects/emo/ under a form of the Mozilla 1.1 open-source license.

## 7.0 Conclusions

As various disciplines attempt to standardize the exchange of information via XML, modeling teams may benefit from a tool that supports collaborative data modeling. EMo is a by-product of an effort by domain experts to collaborate in drafting a coherent data model. It promotes the use of examples, avoiding UML representations and leveraging the experience of team members with Web forms and online information sharing. EMo proved sufficiently malleable within a project attempting to model the psychometrics of assessment design. Indeed, after the modeling phase, our modeling tool continues to serve as a repository of examples. We actively are developing additional editing capabilities and navigational aids while expanding the library of examples in this system. As the model has matured, the model-editing functions are gathering dust, but the system now provides a library of resources as we discuss content and content-creation "wizards" to scaffold interaction with the system.

Developers who prefer or require more formal modeling still may consider a collaborative modeling environment such as EMo because it can provide a hands-on communication conduit with domain experts. At any point in time, the models produced by EMo can be translated into UML or other notations. Such translation could be automated in the future.

## *References*


Arlow, J., & Neustadt, I. (2001). *UML and the unified process: Practical object-oriented analysis and design*. Boston, MA: Addison-Wesley.

Brown, J. S., & Duguid, P. (2000). *The social life of information*. Cambridge, MA: Harvard Business School Press.

Carroll, J. M. (2000). *Making use: Scenario-based design of human-computer interactions*. Cambridge, MA: MIT Press.

Carroll, J. M., & Rosson, M. B. (1992). *Getting around the task-artifact cycle: How to make claims and design by scenario. ACM Transactions on Information Systems, 10*(2), 181-212.

Chen, P. (1976). The entity relationship model—Toward a unified view of data. *ACM Transactions on Database Systems, 1*(1), 9-36.

Chi, M. T. H., Bassok, M., Lewis, M. W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science, 13*, 145-182.

Domingue, J., Motta, E., Buckingham Shum, S. J., Vargas-Vera, M., Kalfoglou, Y., & Farnes, N. (2001). Supporting ontology-driven document enrichment within communities of practice. In *Proceedings of the First International Conference on Knowledge Capture* (pp. 30-37). New York: ACM Press.

Domingue, J. (1998). Tadzebao and WebOnto: Discussing, browsing, and editing ontologies on the Web. In *Proceedings of the 11th Workshop on Knowledge Acquisition, Modeling and Management*. Retrieved May 1, 2004, from http://ksi.cpsc.ucalgary.ca/KAW/KAW98/domingue/

Dreyfus, H. (1993). *What computers still can't do: A critique of artificial reason*. Cambridge, MA: MIT Press.

Farquhar, A., Fikes, R., Pratt, W., & Rice, J. (1995). Collaborative ontology construction for information integration (Technical Report KSL-95-63). Stanford, CA: *Knowledge Systems Laboratory Department of Computer Science*, Stanford University. Retrieved May 1, 2004, from http://www-ksl.stanford.edu/KSL_Abstracts/KSL-95-63.html

Fix Protocol, Ltd. (2004). *Financial Information eXchange ("FIX") protocol.* Retrieved May 1, 2004 from http://www.fixprotocol.org/

Gennari, J. H., Musen, R. W., Fergerson, W. E., Grosso, M. C., Crubézy, M., Eriksson, H., Noy, N. F., & Tu, S. W. (2003). The evolution of Protégé: An environment for knowledge-based systems development. *International Journal of Human-Computer Studies, 58*(1), 89-123.

Gentleware AG. (2004). *Poseidon for UML*. Hamburg, Germany: Author. Retrieved May 1, 2004, from http://www.gentleware.com

Goldfarb, C., & Prescod, P. (1998). *The XML handbook*. Indianapolis, IN: Prentice Hall PTR.

Gruber, T. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies, 43*(5/6), 907-928.

Health Level Seven, Inc. (2004). *HL7 Version 2 messaging standard: Application protocol for electronic data exchange in healthcare environments*. Ann Arbor, MI: Author. Retrieved May 1, 2004, from http://www.hl7.org

Hypersonic SQL Group. (2004). *HSQL database engine project*. Retrieved May 1, 2004, from http://hsqldb.sourceforge.net/

IBM Corporation. (2004). *Rational Rose software*. White Plains, NY: Author. Retrieved May 1, 2004, from http://www-306.ibm.com/software/rational/

IMS Global Learning Consortium, Inc. (2000). *IMS Question & Test Interoperability specification: A review* (White Paper IMSWP-1 Version A). Burlington, MA: Author. Retrieved May 1, 2004, from http://www.imsglobal.org/question/whitepaper.pdf

Jcorporate Ltd. (2004). *Expresso Web Application Development Framework, version 5.0*. Retrieved May 1, 2004, from http://www.jcorporate.com/html/products/expresso.html

LeFevre, J. A., & Dixon, P. (1986). Do written instructions need examples? *Cognition and Instruction, 3*, 1-30.

Microsoft Corporation. (2004). *NetMeeting software*. Redmond, WA: Author. Retrieved May 1, 2004, from http://www.microsoft.com/windows/netmeeting/

Mislevy, R. J., Chudowsky, N., Draney, K., Fried, R., Gaffney, T., Haertel, G., et al. (2003). *Design patterns for assessing science inquiry* (PADI Technical Report 1). Menlo Park, CA: SRI International. Retrieved May 1, 2004, from http://padi.sri.com/downloads/PADI_DesignPatterns.pdf

MySQL, Inc. (2004). *The MySQL Database Server*. Seattle, WA: Author. Retrieved May 1, 2004, from http://www.mysql.com/

Pirolli, P., & Anderson, J. R. (1985). The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology, 39*, 240-272.

Recker, M., & Pirolli, P. (1995). Modelling individual differences in students' learning strategies. *The Journal of the Learning Sciences, 4*, 1-38.

Sanders, G. L. (1995). *Data modeling*. Danvers, MA: Boyd & Fraser Pub. Co.

Schank, P., & Hamel, L. (2004). Collaborative modeling: Hiding UML and promoting data examples in EMo. *Computer Supported Collaborative Work* (Chicago, IL, November 6-11, 2004).

Schuler, D., & Namioka, A. (1993). *Participatory design: Principles and practices*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Suchman, L. (1987). *Plans and situated actions: The problem of human-machine communication*. New York: Cambridge University Press.

Sullivan, K. (2004). *Preliminary report of the NSF Workshop on the Science of Design: Software and Software Intensive Systems*. Retrieved on May 1, 2004, from http://www.cs.virginia.edu/~sullivan/sdsis/SDSIS%20Preliminary%20Report%20020210b.pdf

Sun Microsystems. (2004). *JavaServer Pages technology*. Retrieved May 1, 2004, from http://java.sun.com/products/jsp/

Swartout, B., Patil, R., Knight, K., & Russ, T. (1996). Toward distributed use of large-scale ontologies. In *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop*. Retrieved May 1, 2004, from http://ksi.cpsc.ucalgary.ca/KAW/KAW96/swartout/Banff_96_final_2.html

Tigris.org. (2004). *ArgoUML tool*. Retrieved May 9, 2004, from http://argouml.tigris.org/

Tilley, S., & Huang, S. (2003). A qualitative assessment of the efficacy of UML diagrams as a form of graphical documentation in aiding program understanding. In *Proceedings of the 21st Annual International Conference on Documentation* (pp. 184-191). New York: ACM Press.

U.S. Congress, Office of Technology Assessment. (1992). *Testing in American schools: Asking the right questions* (OTA-SET-519). Washington, DC: U.S. Government Printing Office.

Zhu, X., & Simon, H. A. (1987). Learning mathematics from examples and by doing. *Cognition and Instruction, 4*, 137-166.


**Sponsor**
The National Science Foundation, Grant REC-0129331

**Prime Grantee**
SRI International. *Center for Technology in Learning*

**Subgrantees**
University of Maryland
University of California, Berkeley. *Berkeley Evaluation & Assessment Research (BEAR) Center* and *The Full Option Science System (FOSS)*
University of Michigan. *BioKIDs*

SRI International