Customizable Descriptions of Object-Oriented Models

Benoit Lavoie
CoGenTex, Inc.
840 Hanshaw Road
Ithaca, NY 14850, USA
benoit@cogentex.com

Owen Rambow
CoGenTex, Inc.
840 Hanshaw Road
Ithaca, NY 14850, USA
owen@cogentex.com

Ehud Reiter
Department of Computer Science
University of Aberdeen
Aberdeen AB9 2UE, Scotland
ereiter@csd.abdn.ac.uk

1 Introduction: Object Models

With the emergence of object-oriented technology and user-centered software engineering paradigms, the requirements analysis phase has changed in two important ways: it has become an iterative activity, and it has become more closely linked to the design phase of software engineering (Davis, 1993). A requirements analyst builds a formal object-oriented (OO) domain model. A user (domain expert) validates the domain model. The domain model undergoes subsequent evolution (modification or adjustment) by a (perhaps different) analyst. Finally, the domain model is passed to the designer (system analyst), who refines the model into a OO design model used as the basis for implementation. Thus, we can see that the OO models form the basis of many important flows of information in OO software engineering methodologies. How can this information best be communicated?

It is widely believed that graphical representations are easy to learn and use, both for modeling and for communication among the engineers and domain experts who together develop the OO domain model. This belief is reflected by the large number of graphical OO modeling tools currently in research labs and on the market. However, this belief is not accurate, as some recent empirical studies show. For example, Kim (1990) simulated a modeling task with experienced analysts and a validation task with sophisticated users not familiar with the particular graphical language. Both user groups showed semantic error rates between 25% and 70% for the separately scored areas of entities, attributes, and relations. Relations were particularly troublesome to both analysts and users. Petre (1995) compares diagrams with textual representations of nested conditional structures (which can be compared to OO modeling in the complexity of the "paths" through the system). She finds that "the intrinsic difficulty of the graphics mode was the strongest effect observed" (p.35). We therefore conclude that graphics, in order to assure maximum communicative efficiency, needs to be complemented by an alternate view of the data. We claim that the alternate view should be provided by an explanation tool that represents the data in the form of a fluent English text. This paper presents such a tool, the ModelExplainer, or ModEx for short, and focuses on the customizability of the system.[1]

Automatically generating natural-language descriptions of software models and specifications is not a new idea. The first such system was Swartout's GIST Paraphraser (Swartout, 1982). More recent projects include the paraphraser in ARIES (Johnson et al., 1992); the GEMA data-flow diagram describer (Scott and de Souza, 1989); and Gulla's paraphraser for the PPP system (Gulla, 1993). ModEx certainly belongs in the tradition of these specification paraphrasers, but the combination of features that we will describe in the next section (and in particular the customizability) is, to our knowledge, unique.

2 Features of ModEx

ModEx was developed in conjunction with Andersen Consulting, a large systems consulting company, and the Software Engineering Laboratory at the Electronic Systems Division of Raytheon, a large Government contractor. Our design is based on initial interviews with software engineers working on a project at Raytheon, and was modified in response to feedback during iterative prototyping when these software engineers were using our system.

• ModEx output integrates tables, text generated automatically, and text entered freely by the user. Automatically generated text includes paragraphs describing the relations between classes, and para-

[1](Lavoie et al., 1996) focuses on an earlier version of ModEx which did not yet include customization.

graphs describing examples. The human-authored text can capture information not deducible from the model (such as high-level descriptions of purpose associated with the classes).

• ModEx lets the user customize the text plans at run-time, so that the text can reflect individual user or organizational preferences regarding the content and/or layout of the output.

• ModEx uses an interactive hypertext interface (based on standard HTML-based WWW technology) to allow users to browse through the model.

• Input to ModEx is based on the ODL standard developed by the Object Database Management Group (Cattell, 1994). This allows for integration with most existing commercial off the shelf OO modeling tools. Some previous systems have paraphrased complex modeling languages that are not widely used outside the research community (GIST, PPP).

• ModEx does not have access to knowledge about the domain of the OO model (beyond the OO model itself) and is therefore portable to new domains.

3 A ModEx Scenario

Suppose that a university has hired a consulting company to build an information system for its administration. Figure 1 shows a sample object model for the university domain (adapted from (Cattell, 1994, p.56), using the notation for cardinality of Martin and Odell (1992)) that could be designed by a requirements analyst.

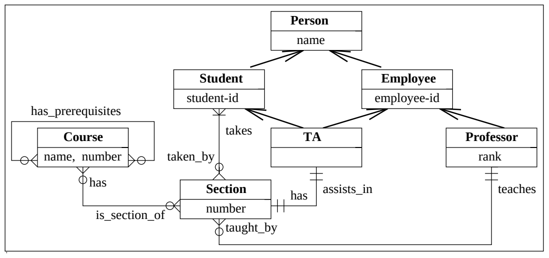
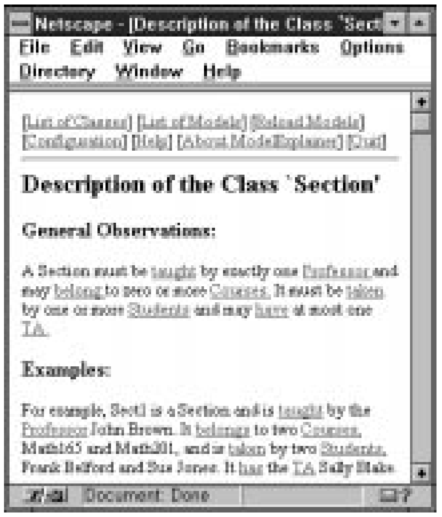


Figure 1: The University O-O Diagram

Once the object model is specified, the analyst must validate her model with a university administrator (and maybe other university personnel, such as data-entry clerks); as domain expert, the university administrator may find semantic errors undetected by the analyst. However, he is unfamiliar with the "crow's foot" notation used in Figure 1. Instead, he uses ModEx to generate fluent English descriptions of the model, which uses the domain terms from the model. Figure 2 shows an example of a description generated by ModEx for the university model. Suppose that in browsing through the model using the hypertext interface, the university administrator notices that the model allows a section to belong to zero courses, which is in fact not the case at his university. He points out the error to the analyst, who can change the model.

Suppose now that the administrator finds the texts useful but insufficient. To change the content of the output texts, he can go to the Text Plan Configuration window for the text he has been looking at, shown in Figure 3. He can add to the text plan specification one or more constituents (paragraphs) from the list of pre-built constituents (shown in the lower right corner of Figure 3). After saving his modifications, he can return to browsing the model and obtain texts with his new specifications.


Netscape - [Description of the Class 'Sect
File Edit View Go Bookmarks Options
Directory Window Help
[List of Classes] [List of Models] [Reload Models]
[Configuration] [Help] [About ModelExplainer] [Quit]
Document: Done

Description of the Class 'Section'

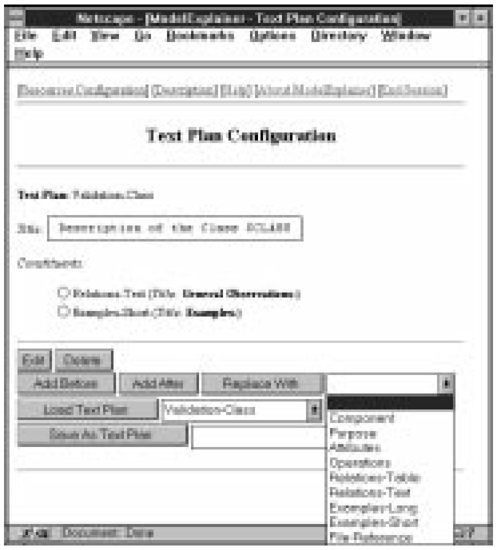
General Observations:

A Section must be taught by exactly one Professor and may belong to zero or more Courses. It must be taken by one or more Students and may have at most one TA.

Examples:

For example, Sect1 is a Section and is taught by the Professor John Brown. It belongs to two Courses, Math165 and Math201, and is taken by two Students, Frank Belford and Sue Jones. It has the TA Sally Blake.

Figure 2: Description Used for Validation



Text Plan Configuration

Text Plan: Validation-Class

Title: Description of the Class $CLASS

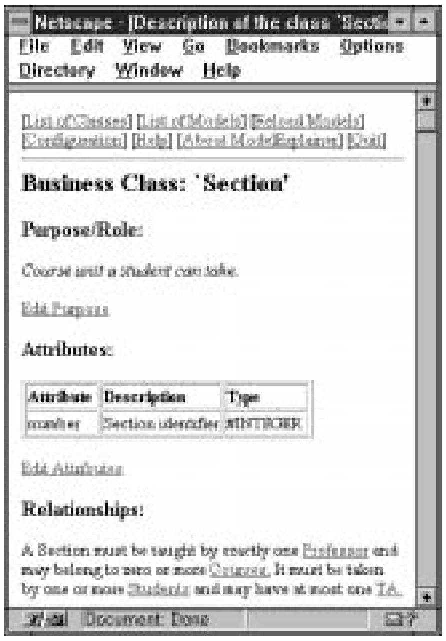
Constituents:

○ Relations-Text (Title: **General Observations**)
○ Examples-Short (Title: **Examples**)

Validation-Class

Figure 3: Text Plan Configuration Interface

Once the model has been validated by the univer-


Netscape - [Description of the class 'Secti
File Edit View Go Bookmarks Options
Directory Window Help
Document: Done

[List of Classes] [List of Models] [Reload Models] [Configuration] [Help] [About ModelExplainer] [Quit]

Business Class: 'Section'

Purpose/Role:

Course unit a student can take.

Edit Purpose

Attributes:

Attribute	Description	Type
number	Section identifier	#INTEGER

Edit Attributes

Relationships:

A Section must be taught by exactly one Professor and may belong to zero or more Courses. It must be taken by one or more Students and may have at most one TA.

Figure 4: Description Used for Documentation

sity administrator, the analyst needs to document it, including annotations about the purpose and rationale of classes and attributes. To document it, she configures an output text type whose content and structure is compatible with her company's standard for OO documentation. An example of a description obtained in modifying the text plan of Figure 3 is shown in Figure 4. (This description follows a format close to Andersen Consulting's standard for documentation.) This description is composed of different types of information: text generated automatically (section *Relationships*), text entered manually by the analyst because the information required is not retrievable from the CASE tool object model (section *Purpose*), and tables composed both of information generated automatically and information entered manually (section *Attributes*). The analyst then saves the text plan under a new name to use it subsequently for documentation purposes. Note that while the generated documentation is in hypertext format and can be browsed interactively (as in the I-DOC system of Johnson and Erdem (1995)), it can of course also be printed for traditional paper-based documentation and/or exported to desktop publishing environments.

4 How ModEx Works

As mentioned above, ModEx has been developed as a WWW application; this gives the system a platform-independent hypertext interface. Figure 5 shows the ModEx architecture. ModEx runs as a server which receives requests via a standard Web CGI interface and returns HTML-formatted documents which can be displayed by any standard Web browser. The documents generated by ModEx are always generated dynamically in response to a request, and are composed of human-authored text, generated text and/or generated tables. The main requests are the following:

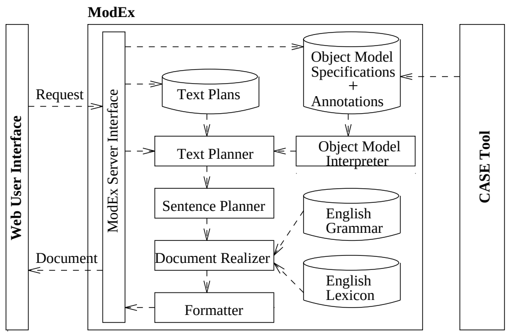


Figure 5: ModEx Server Architecture

• *Text Plan Editing.* This generates an HTML document such as that shown in Figure 3 which allows a user to load/edit/save a text plan macro-structure specification. A representation corresponding to the text plan of Figure 3 is shown in Figure 6. Once edited, this representation can be stored permanently in the library of text plans and can be used to generate descriptions. In this representation, *User Text* indicates free text entered for a title, while *Relations-Text* and *Examples-Short* are schema names referring to two of the eight predefined text functions found in a C++ class library supplied with ModEx.

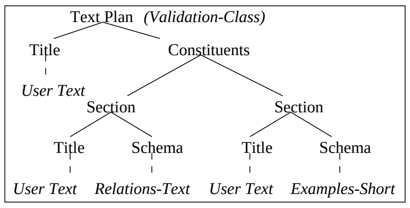


Figure 6: Macro-Stucture for Text Plan of Figure 3

• *Object Model Loading.* This loads an object model specification and generates a document displaying the list of classes found in the model.

• *Description Generation.* This returns a description such as that shown in Figures 2 or 4. To generate a description, the text planner creates a text structure corresponding to the text plan configuration selected by the user. This text structure is a constituency tree where the internal nodes define the text organization, while the bottom nodes define its content. The text content can be specified as syntactic representations, as table specification and/or as human-authored text for the titles and the object model annotations. The text structure is transformed by the sentence planner which can aggregate the syntactic representations (cf. conjunctions *and* in description on Figure 2) or introduce cue words between constituents (cf. expression *For example* on Figure 2). The resulting text structure is then passed to the text realizer which uses RealPro (Lavoie and Rambow, 1997), a sentence realizer, to realize each individual syntactic representation in the text structure. Finally, a formatter takes the final text structure to produce an HTML document.

• *Object Model Annotation Editing.* This allows the user to edit human-authored annotations of the object model. This editing can be done via links labelled *Edit ...* which appear in Figure 4. These human-authored texts are used by some of the predefined text functions to generate the descriptions.

5 Outlook

ModEx is implemented in C++ on both UNIX and PC platforms. It has been integrated with two object-oriented modeling environments, the ADM (Advanced Development Model) of the KBSA (Knowledge-Based Software Assistant) (Benner, 1996), and with Ptech, a commercial off-the-shelf object modeling tool. ModEx has been fielded at a software engineering lab at Raytheon, Inc.

The evaluation of ModEx is based on anecdotal user feedback obtained during iterative prototyping. This feedback showed us that the preferences regarding the content of a description can vary depending on the organization (or type of user). The control that ModEx gives over the text macro-structure is one step toward satisfying different types of text requirements. We are currently extending ModEx in order to give the user a better control over the text micro-structure, by replacing the set of predefined C++ text functions with customizable ASCII specifications. This feature should make ModEx more easely portable among different types of users. In addition, we intend to port ModEx to at least two new OO modeling environments in the near future.

Acknowledgments

The first version of ModEx for ADM was supported by USAF Rome Laboratory under contract F30602-92-C-0015. General enhancements to the linguistic machinery were supported by SBIR F30602-92-C-0124, awarded by USAF Rome Laboratory. Current work on ModEx is supported by the TRP-ROAD cooperative agreement F30602-95-2-0005 with the sponsorship of DARPA and Rome Laboratory. We are thankful to K. Benner, M. DeBellis, J. Silver and S. Sparks of Andersen Consulting, and to F. Ahmed and B. Bussiere of Raytheon Inc., for their comments and suggestions made during the development of ModEx. We also thank T. Caldwell, R. Kittredge, T. Korelsky, D. McCullough, A. Nasr and M. White for their comments and criticism of ModEx.

References

Benner, K. (1996). Addressing complexity, coordination, and automation in software development with the KBSA/ADM. In *Proceedings of the Eleventh Knowledge-Based Software Engineering Conference (KBSE-96)*, Syracuse, NY.

Cattell, R. G. G., editor (1994). *The Object Database Standard: ODMG-93.* Morgan Kaufman Publishers, San Mateo, CA.

Davis, A. M. (1993). *Software Requirements.* Prentice-Hall, Inc., Upper Saddle River, NJ, revision edition.

Gulla, J. (1993). *Explanation Generation in Information Systems Engineering.* PhD thesis, Norwegian Institute of Technology.

Johnson, W. L. and Erdem, A. (1995). Interactive explanation of software systems. In *Proceedings of the Tenth Knowledge-Based Software Engineering Conference (KBSE-95)*, pages 155–164, Boston, Mass.

Johnson, W. L., Feather, M. S., and Harris, D. R. (1992). Representation and presentation of requirements knowledge. *IEEE Transactions on Software Engineering*, pages 853–869.

Kim, Y.-G. (1990). *Effects of Conceptual Data Modeling Formalisms on User Validation and Analyst Modeling of Information Requirements.* PhD thesis, University of Minnesota.

Lavoie, B., Rambow, O. and Reiter, E. (1996). The ModelExplainer. In *Demonstration Notes of International Natural Language Generation Workshop (INLG-96)*, Hermonceux Castle, Sussex, UK.

Lavoie, B. and Rambow, O. (1997). A Fast and Portable Realizer for Text Generation Systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, Washinghton,DC..

Martin, J. and Odell, J. (1992). *Object-Oriented Analysis and Design.* Prentice Hall, Englewood Cliffs, NJ.

Petre, M. (1995). Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM*, 38(6):33–42.

Scott, D. and de Souza, C. (1989). Conciliatory planning for extended descriptive texts. Technical Report 2822, Philips Research Laboratory, Redhill, UK.

Swartout, B. (1982). GIST English generator. In *Proceedings of the National Conference on Artificial Intelligence.* AAAI.