

# The MODELEXPLAINER

**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, evolutionary software engineering paradigms, the requirements gathering phase has become an iterative activity. A requirements analyst builds a formal object-oriented (OO) data model (*modeling*). A user (domain expert) performs a *validation* of the formal model. Then, the requirements model undergoes subsequent *evolution* (modification or adjustment) by a (perhaps different) analyst.

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 data model. This belief is reflected by the large number of graphical OO data modeling tools currently in research labs and on the market. However, this belief is a fallacy, 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. (?) compares diagrams with textual representations of nested conditional structures (which can be compared to data modeling in the complexity of the “paths” through the system). He 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 standard English text. This paper presents such a tool, the MODELEXPLAINER, or MODEX for short.

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 is, to our knowledge, unique.

## 2 Features of MODEX

Our design is based on initial interviews with potential users, and on subsequent feedback from actual users during an iterative prototyping approach.

- MODEX includes examples in its texts, as well as conventional descriptions. The need for examples in documentation has been pointed out in recent work by Paris and Mittal (see for example (?) and the references cited therein). However, none of the specification paraphrasers proposed to date have included examples.

- MODEX uses an interactive hypertext interface to allow users to browse through the model.

Such interfaces have been used in other NLG applications, (e.g., (?; ?)), but ours is based on (now) standard html-based WWW technology.

- MODEx uses a simple modeling language, which is based on the ODL standard developed by the Object Database Management Group (OMG) (Cattell, 1994). 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 data model (beyond the data model itself). At least one previous system has used such knowledge (GEMA).

### 3 A MODEx Scenario

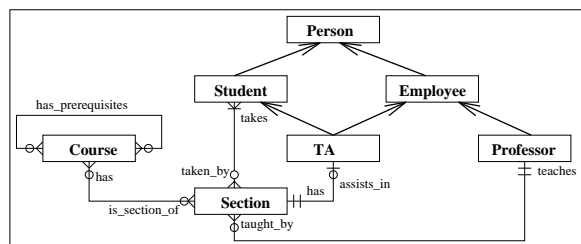


Figure 1: The University O-O Diagram

Suppose that a university has hired a consultant analyst to build an information system for its administration. The analyst has devised a data-model and shows it to a university administrator for validation. The model is shown in Figure 1; it is adapted from (Cattell, 1994, p.56). It uses the “crow’s foot” notation of Martin and Odell (1992) for cardinality on relations. The administrator is not familiar with this notation and cannot easily understand it. He invokes MODEx to generate a textual description in English of a particular aspect of the model, namely of the SECTION class (Figure 2). The text is viewed via a World-Wide-Web browser such as Netscape or Mosaic. The *General Observations* section paraphrases the class definition, and the *Examples* section gives a concrete example of an instance of this class. Hypertext links are included (shown underlined); for example, clicking on *Professor* will produce a description of the PROFESSOR class.

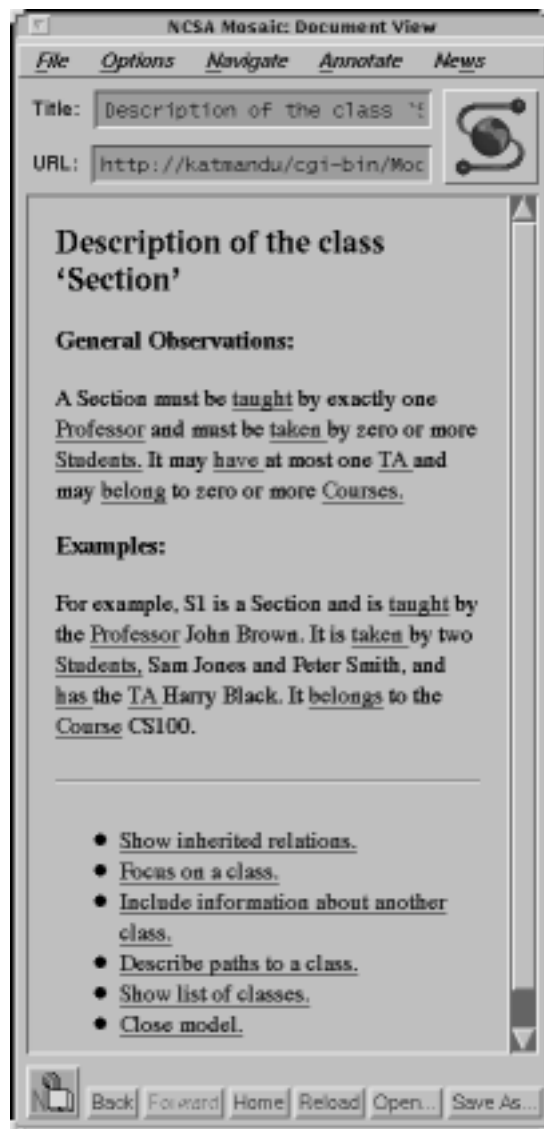


Figure 2: Description of SECTION

Several control buttons give access to additional texts.

The administrator thinks it is strange that a SECTION may belong to zero or more COURSES. He clicks on the word *belong* and obtains the text shown in Figure 3 (top). This text, especially with its boundary-value examples, makes it very clear that the model allows a SECTION to belong to no COURSES, and also allows a SECTION to belong to more than one COURSE. In his institution, each section belongs to exactly one course. (We have observed such cardinality mistakes in many OO models.) The ana-

**Description of the relation ‘Is section of’  
General Observations:**

A section may belong to zero or more Courses. For example, S1 is a Section and belongs to the Course CS100. S2 is a Section and does not belong to any Courses. S3 is a Section and belongs to three Courses, Math100, Physics100, and Eng100.

**Description of the relation ‘Is section of’  
General Observations:**

A section must belong to exactly one Course. For example, S1 is a Section and belongs to the Course CS100.

**Cardinality:**

It is illegal for a Section to belong to zero Courses. For example, it would be illegal for the Section S2 not to belong to any Courses. In addition, It is illegal for a Section to belong to more than one Course. For example, it would be illegal for the Section S3 to belong to two Courses, Math100 and Physics100.

Figure 3: Two descriptions of **is section of**

lyst fixes this and reruns MODEx on this relation, obtaining the description shown in Figure 3 (bottom). The text now contains a new section with negative examples, which makes it clear that it is no longer possible for a SECTION to belong either to zero COURSES or to multiple COURSES.

Several other types of text can be generated, such as path descriptions and comparisons and texts about several classes. We refer to (?) for more detailed information.

## 4 How MODEx Works

MODEx is implemented using the now fairly standard, modular pipeline architecture. Several modules are part of COGENT, CoGenTex’s generator shell. MODEx operates as a ‘Web server’ which generates HTML files that can be viewed by any Web browser. For lack of space we refrain from giving details here and refer to (?) for details.

## 5 Restrictions on the Object Model

MODEx is designed for use independent of the domain of the OO data model that is being described: it lacks domain knowledge. This means

that the system is fully portable between modeling domains, and is not overly costly to use. However, this also means that the system cannot detect semantic modeling errors. Instead, MODEx works by providing the analyst or domain expert with a different representation of the model, namely in English. Having a second view in an easily accessible code allows him or her to more easily detect semantic errors.

Furthermore, the lack of domain knowledge also means that MODEx cannot choose the correct paraphrase for an ambiguous part of a model. For example, analysts usually label relations with either nouns or verbs, giving rise to paraphrases such as *A committee determines issues* (verb) or *A committee has an issue as its topic* (noun). However, suppose the analyst introduces a relation called **top** between classes GULFINKEL and WORROW. Since *top* can be either a noun or a verb in English, the analyst could either mean that *A gulfinkel tops a worrow* or that *A gulfinkel has a worrow as its top*. The two statements are presumably incompatible, but the correct one can only be chosen on the basis of knowledge about the (fictitious) gulfinkel-worrow domain – which MODEx lacks.

We deal with this problem by requiring the MODEx user to follow certain conventions with respect to the labeling of relations and objects. The MODEx expects classes to be labeled with singular nouns, and relations to be labeled with third person singular active verbs, passive verbs with *by*, or nouns. Verbs and nouns can be followed by a preposition, and there can be additional material (arguments, adjuncts) between a verb and its preposition.

In fact, while such conventions appear to be limiting at first, they serve a second purpose, namely that of imposing discipline in naming. In a data model, it is important that names be used consistently for naming objects and relations, since otherwise the model is difficult to understand whether or not MODEx is used. For example, a designer, looking at a graphical representation of a data model, may well misunderstand the gulfinkel-worrow relation above and interpret it in the opposite manner from what the re-

quirements analyst (who devised the model) intended. Larger object-oriented software engineering projects therefore develop naming conventions, and Martin and Odell (1992, p.134) say (somewhat vaguely) that two classes connected by a relation “ought to read like a sentence”. Thus, MODEx can serve the purpose of enforcing such naming conventions, since if they are not followed, the text will be nonsensical, or even unreadable.

## 6 Outlook

The 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) (?), and with Ptech, a commercial off-the-shelf object modeling tool. MODEx has been fielded at a software engineering lab at Raytheon, Inc., with interesting and encouraging initial feedback.

Currently, we are pursuing several development directions. For example, we are extending the system to allow the user to enter free text associated with particular objects in the model (such as classes, attributes). This free text can capture information not deducible from the model (such as high-level descriptions of purpose), and will be integrated with the automatically generated text. We are also developing a facility to direct the output of MODEx to commercial off-the-shelf publishing environments for the production of standard (paper-based) documentation.

## Acknowledgments

Initial development of MODEx was funded by USAF Rome Laboratory under contracts F30602-92-C-0015 and F30602-92-C-0124. 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 thank F. Ahmed, K. Benner, B. Bussière, M. DeBellis, J. Silver, and S. Sparks for their comments and suggestions, and T. Caldwell, R. Kittredge, T. Korelsky, D.

McCullough and M. White and two anonymous reviewers for their comments and criticism of MODEx and the present paper.