Electronic Communications of the EASST
Volume 34 (2010)

Proceedings of the 6th Educators' Symposium: Software Modeling in Education at MODELS 2010 (EduSymp 2010)

Position Paper: m2n—A Tool for Translating Models to Natural Language Descriptions

Petra Brosch and Andrea Randak

8 pages

Guest Editors: Peter J. Clarke, Martina Seidl
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Position Paper: m2n—A Tool for Translating Models to Natural Language Descriptions

Petra Brosch* **and Andrea Randak**

lastname@big.tuwien.ac.at, http://www.big.tuwien.ac.at
Business Informatics Group
Vienna University of Technology, Austria

Abstract: To describe the structure of a system, the UML Class Diagram yields the means-of-choice. Therefor, the Class Diagram provides concepts like class, attribute, operation, association, generalization, aggregation, enumeration, etc. When students are introduced to this diagram, they often have to solve exercises where texts in natural language are given and they have to model the described systems. When analyzing such exercises, it becomes evident that certain kinds of phrases describing a particular concept appear again and again contextualized to the described domain.

In this paper, we present an approach which allows the automatic generation of textual specifications from a given Class Diagram based on standard phrases in natural language. Besides supporting teachers in preparing exercises, such an approach is also valuable for various e-learning scenarios.

Keywords: modeling exercises, natural language description

1 Introduction

When teaching modeling, one of the most repetitive and time consuming tasks is the development of exercises and the corresponding solutions. A typical exercise consists of a given textual description of an arbitrary domain (e.g., university systems, enterprises, airports) which students have to model for example with a UML Class Diagram. In order to obtain an adequate exercise, it does not suffice to prepare the textual specification only, but also the sample solution has to be modeled for checking if all taught concepts are covered and if the difficulty level and size are reasonable. Hence, the teacher has to describe the same content twice: once as textual specification in natural language and once as a model. As there should be a one-to-one correspondence between text and model, the question at hand is if it is possible to automatically derive one artifact from the other.

Already in the early 1980s attempts of automated translation of textual descriptions into program code were conducted. R.J. Abbott [Abb83] discusses a method for deriving programming concepts like data types and references by analyzing informal English descriptions. An important remark of Abbott's work relates to the automation level of such a method. He points out that such a transformation is far away from being fully automated. User interaction is still needed

* Funding for this research was provided by the fFORTE WIT - Women in Technology Program of the Vienna University of Technology, and the Austrian Federal Ministry of Science and Research.

to make the outcome perfect. After all natural language is imprecise and therefore leaves room for interpretation. Only if the text obeys a certain structure and the sentences are expressed in a precise, unambiguous way, the models can be automatically derived as it is for example done in the field of requirements engineering (cf. [FKM05, WKH08]).

In contrast to models of real-world software engineering projects, the models of the exercises have not to express customers' and users' expectations, but they have to fulfill certain didactical expectations and usually show fictive, simplified scenarios only. Therefore, the construction is different: when a teacher prepares an exercise, (s)he usually does not write the text and model the sample solution afterwards, but starts with modeling the sample solution. Having the sample solution at hand, a formal specification of the scenario is available. In this paper, we propose to use the sample solution for generating the natural language textual description of the exercise.

2 Background

With five years experience in teaching the course *Object-Oriented Modeling* at the Vienna University of Technology [BSW+08], it becomes evident that specific modeling concepts are expressed by recurring phrases, solely contextualized to the described domain. Phrases like "part of" for expressing composite aggregations or "is a" for generalizations are used repeatedly, irrespective of describing houses, persons, or elements of any other domain.

Fig. 1 shows a typical toy example of our modeling course, covering the basic concepts of Class Diagrams. One possible textual description is as follows.

> *Persons have a name. Guides and visitors are persons. A guide leads multiple guided tours. Each guided tour has exactly one guide. Each visitor may attend multiple guided tours, guided tours are attended by one to 20 visitors. For each guided tour the id and the duration are known. A guided tour covers exactly one sight, but each sight is covered by multiple guided tours. A sight has a name and an address. A sight is located in one city. A city may have multiple sights. For each city name and size are known.*

Although this textual description sounds natural, it follows an algorithmic pattern. In a first step, the most important class of the model is identified as a starting point for the description and input for the algorithm. This class is described by its name and its attributes. Then, inheritance relations and associations follow. In the order of exploring related classes, the procedure recurs.

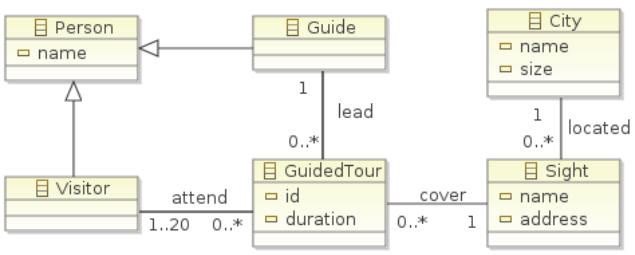

Figure 1: Class Diagram example

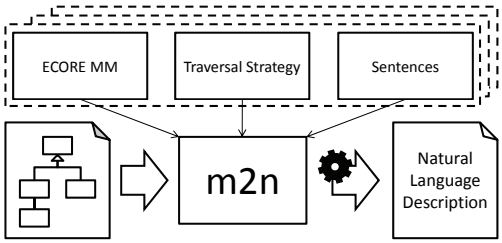

Figure 2: m2n Architecture

3 Realization

The m2n tool (short for model to natural language description) yields a framework for translating models of arbitrary metamodels to natural language descriptions as depicted in Fig. 2. The implementation is based on the Eclipse Modeling Framework[1], hence any ECORE based metamodel may be integrated. Three artifacts are necessary to define a translation specification for a modeling language: a metamodel, a model traversal strategy, and a set of sentence templates for describing specific concepts.

In the following, these artifacts are examined in detail for a simplified Class Diagram.

Metamodel. The Class Diagram metamodel used in our first prototype comprises a restricted set of the UML Class Diagram concepts. At the moment, we support the concepts of Class, Attribute, Generalization (single inheritance), and Association (binary).

Sentence Templates. Standard phrases in the form of sentence templates have to be provided for each concept of the metamodel. If there are more than one sentence for a concept, one is randomly selected, which makes the generated text more natural. These sentences contain wildcards, which are replaced by concrete model values.

Traversal Strategy. Each metamodel needs a dedicated traversal strategy to explore the model. A traversal strategy should implement a dedicated interface to allow reliable integration into the m2n framework by the dynamic class loading mechanism of Java.

The traversal strategy for the Class Diagram implements a special kind of a breadth first search as shown in Listing 1. The most important model element is identified by a heuristic and acts as a starting point for the algorithm. Currently, subclass relations and associations are counted. Generalizations are higher ranked than associations. In the example of Fig. 1, the class Person is selected as most important element, because of its two subclasses.

For the actual text generation, the root class is first introduced in the generated text by a sentence describing its name and attributes. Second, all direct subclasses are named and put to a queue holding the succeeding model elements to describe, if they are not printed so far. Third, associations are specified. Currently, the reading direction is not explicitly available in the Class Diagram metamodel. To construct a sentence, the direction is derived from source and target

[1] http://www.eclipse.org/modeling/emf/

roles of the association. Referenced classes are then put into the queue. As long as the queue is not empty, the algorithm prints the details for the next node. If the model is split into parts, then a new root node has to be found. The algorithm terminates, when all model elements are printed.

```
1  /* variable declarations */
2  Queue nodeQueue; /* temporary buffer for all distinct visited elements */
3  Set printedME;   /* holds already printed model elements */
4  Set allME;       /* holds all model elements */
5
6  /* function declarations */
7  void getText() { /* implemented method of interface; called by m2n */
8    while(printedME != allME) {   /* all model elements printed? */
9      node = localizeRootClass(); /* search for most important model element */
10     nodeQueue.add(node);        /* add node as starting class to nodeQueue */
11     printModel();               /* print model details */
12   }
13 }
14
15 void printModel() {
16   node = nodeQueue.poll();   /* get node to describe */
17   printAttributes(node);     /* print sentence for attributes */
18   printInheritance(node);    /* print sentence for inheritance;
19                                 add each new subclass to nodeQueue */
20   printAssociation(node);    /* print sentence for association;
21                                 add each new associated class to nodeQueue */
22   printedME.add(node);       /* printing of node completed */
23   if (!nodeQueue.isEmpty()) /* if any nodes left, repeat; */
24     printModel();
25 }
```

Listing 1: Model to Natural Language Description Generation

4 The Long-Term Vision

The presented m2n-tool supports on the one hand teachers in keeping exercises consistent with the sample solutions, i.e., it facilitates the management of teaching artifacts. On the other hand, if provided to the students, they can generate natural text out of the given diagram, providing them some explanation of the diagram.

But these are only side-effects; we developed our tool with a very different intention in mind. m2n is intended to be used within a major e-learning project aiming at the development of an interactive e-learning system for Class Diagrams. In the following, we motivate why we need such an e-learning system, then we shortly describe the basic architecture, and finally we explain the important role m2n plays in these considerations.

4.1 Motivation

One of the basic challenges in teaching modeling is the practical part of the course. Unlike in programming where the programs of the students are relatively easily testable by checking whether they show the expected behavior or not, the situation in modeling is different. Especially

when models are used for describing, analyzing, or designing systems, and when the models are not executable or transferable to a formal specification like code, automatic testing is hardly possible. A serious correction and grading is only possible by human teachers, often requiring considerable intellectual effort to follow the students' approach, because students often have a different viewpoint on a matter, which is still correct. It is also possible, that the specification is interpretable in different ways.

Especially in teaching modeling, learning by doing is extremely important, so such practical exercises in which the students have to derive a model from a textual specification. Even if the students use modeling tools instead of drawing the models by hand, automatic correction is hardly possible, and if it is possible then the specification has to be formulated very narrow, teaching the students to recognize patterns only instead of allowing them to be creative.

Due to this reasons it is extremely difficult, to build e-learning environments which allow the students to train on their own and to get valuable feedback from the system, especially when no teacher is at hand. Naturally, the exercises may be formulated in a very static manner in terms of multiple choice questions, where the students have to decide if a statement about a given model is true or false. Another approach is to give the students textual specifications and a sample solution, which they can compare to their own result. Although such approaches are certainly helpful, they do not allow to train creativity, which is certainly an important skill in practical modeling. In the following, we propose an approach which allows a more flexible way to practice the modeling of Class Diagrams.

4.2 The Basic Architecture

The ideal case for teaching modeling is that the students have to realize a software engineering project where they have to create the models as primary design and documentation artifacts. Since in many courses it is not possible to combine teaching modeling with realizing a practical project (or if the assumption is that it is preferable to learn the necessary concepts first before putting them into practice), typical modeling exercises consist of small textual specifications as they may be created with m2n. The students are asked to model the given text as precisely as possible. A good approach is as follows:

1. identify classes,
2. decorate classes with attributes and operations,
3. include inheritance relationships,
4. introduce the associations between classes.

One of the major problems hampering the automatic correction is the naming of the modeling elements. Model elements are usually identified by their names and often there is a huge choice of freedom on how to name an element. To circumvent this problem we propose the following approach.

The specification is internally annotated with the information which words describe model elements. Considering our example from above, the annotated specification contains information

that a Guide is a class, that Person is a class and that name is an attribute of Person. Naturally, the annotations are not visible to the students, as they contain the solutions. Additionally, the relationships between the model elements are known.

In the proposed tool, the student only sees the given textual specification. First, (s)he is allowed to select those terms of the text which (s)he thinks represents classes. Having identified the classes, attributes and operations are added similarly. When the student selects a term in the text, a corresponding model element is created in a graphical editor. When the student is done with the identification of model elements, (s)he adds the relationships between the model elements. The system monitors the student's behavior and is able to give immediate feedback.

Similar approaches have been presented in [APL08, BM06, SM04], but our approach is novel in the way how the specifications are created.

4.3 The Role of m2n

Whenever a new exercise is created, the specification has to be prepared accordingly in order to be suitable for being used as input for the e-learning tool. Without automatic support, the creation of the specification with the required information is cumbersome and time-consuming. Furthermore, it has to be done extremely carefully because otherwise the exercise in the e-learning system is buggy and the students either get confused or they even learn wrong things.

For supporting the teachers in easily creating new exercises, we propose an extension of m2n which is able to create such an annotated specification automatically which may serve as input for the e-learning tool. As we have already seen, the teacher provides the sample solution model and m2n creates the textual version. Furthermore, the extension of m2n also creates an annotated version of the specification which may be directly imported from the e-learning tool. Naturally, the sample solution model provides all information necessary and the required annotation is straightforward in the case that the text is used as provided by our tool. If manual postprocessing is desired, then the danger is that the model and specification are not synchronized anymore. It may be possible that certain features are not derived as expected and then manual intervention is necessary. For this problem, we intend to develop a sophisticated update mechanism allowing to propagate modifications not only from the model to the specification but also vice versa, i.e., from the specification to the model.

Having this extension at hand, we expect to offer an e-learning tool for Class Diagrams which allows an easy creation of exercises with minimizing the problem of inconsistencies between specification and model.

5 Conclusion

In this paper, we presented a tool for transforming a given Class Diagram to a textual specification in natural language. With such a tool teachers can automatically create the textual specification from the sample solution of an exercise which may serve as a basis for the description of the students' homework or for examination questions.

We realized a first prototype implementation as Eclipse Plugin. Although the text generated in first experiments yield a good basis for describing a given model, the implementation leaves

much room for interesting extensions and fine-tuning. For example, currently the plural of a noun is obtained by appending an "s" at the end of the word—irregular forms are neglected. Furthermore, we will extend the collection of sentences and elaborate on a more sophisticated assembly algorithm of the text in order to obtain a more natural specification. Also we consider a subset of the Class Diagram's elements only at the moment—e.g., we are not able to express association classes or n-ary associations which may be treated as any other model element. Furthermore, better synchronization support between text and model will be an issue, because if the text has been edited manually and the model is modified, then the manual changes of the text should not get lost.

Our tool may also be used by the students for practicing. Students get a textual description of their model and by experimenting they obtain an explanation of the modifications' impact. For the realization of this use case, it will be necessary to build a dedicated user interface which is able to highlight the modifications.

The long-term goal is building an e-learning framework for learning UML diagrams. Given a textual specification, the students shall identify model elements like classes, associations, aggregations etc. which have to be arranged as described in the specification. The result of the students' effort is compared to a sample solution and differences (i.e., the mistakes) are reported. Similar approaches are presented in [APL08, BM06, SM04]. In order to obtain the link between textual specification and sample solution, the text has to be annotated, which is done by hand so far. With the approach presented in this paper it will be possible to annotate the textual specification automatically, facilitating the creation of new exercises.

Bibliography

[Abb83] R. J. Abbott. Program Design by Informal English Descriptions. *Communications of the ACM* 26(11):882–894, 1983.

[APL08] L. Auxepaules, D. Py, T. Lemeunier. A Diagnosis Method that Matches Class Diagrams in a Learning Environment for Object-Oriented Modeling. In *ICALT '08: Proc. of the 2008 Eighth IEEE International Conference on Advanced Learning Technologies*. Pp. 26–30. IEEE Computer Society, 2008.

[BM06] N. Baghaei, A. Mitrovic. A Constraint-Based Collaborative Environment for Learning UML Class Diagrams. In *Proc. Intelligent Tutoring Systems*. LNCS 4053, pp. 176–186. Springer, 2006.

[BSW+08] M. Brandsteidl, M. Seidl, M. Wimmer, C. Huemer, G. Kappel. Teaching Models @ BIG - How to Give 1000 Students an Understanding of the UML. In *Promoting Software Modeling Through Active Education, Educators Symposium Models'08*. Pp. 64–68. Warsaw University of Technology, 2008.

[FKM05] G. Fliedl, C. Kop, H. C. Mayr. From Textual Scenarios to a Conceptual Schema. *Data & Knowledge Engineering* 55(1):20–37, 2005.

[SM04] P. Suraweera, A. Mitrovic. An Intelligent Tutoring System for Entity Relationship Modelling. *International Journal of Artificial Intelligence in Education (IJAIED)* 14(3-4):375–417, 2004.

[WKH08] K. Wolter, T. Krebs, L. Hotz. A Combined Similarity Measure for Determining Similarity of Model-based and Descriptive Requirements. In *Artificial Intelligence Techniques in Software Engineering (ECAI 2008 Workshop)*. Pp. 11–15. 2008.