Tuna: Ontology-Based Source Code Navigation and Annotation

Christian Zimmer
zimmera@cip.ifi.lmu.de
Klinikweg 16
85643 Steinhöring

Axel Rauschmayer*
Axel.Rauschmayer@ifi.lmu.de
Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstr. 67
D-80538 München, Germany

Abstract

Through the constant growth of the size and complexity of software systems, software engineering is—now more than ever—a challenging field. To help the developers, new development processed have been invented: the Model-Driven Architecture (MDA) and Extreme Programming (XP). Both turn to modeling to raise the level of abstraction and to increase programmer efficiency. MDA could be seen as an inverse of XP: where MDA uses the model of the system to generate code, the XP developer codes almost from the start. The source code is then supposed to be expressive enough to make it obvious what the model is. Both methods have their disadvantages. The generation process of MDA is usually not flexible enough, while in XP, the source code is insufficiently expressive for representing all modeling knowledge. The goal of our tool TUNA is to move XP closer to MDA by giving it rich means for integrating modeling concepts with the source code. To that end, we created a generic source code ontology and represent its instances as *topic maps* [10]. Additionally, topic maps enable TUNA to implement a powerful query mechanism.

1 Introduction

The size of software systems and therefore their complexity is constantly growing. This complicates maintainability and extensibility for the engineer. To meet this challenge, new software engineering processes have been created. Two of the most recent additions to this field are the Model Driven Architecture (MDA,[9]) and Extreme Programming (XP,[2]). An MDA developer models the whole system in the Unified Modeling Language (UML) first. He uses iterating

*Supported by Deutsche Forschungsgemeinschaft (DFG) project WI 841/6-1 "InOpSys"

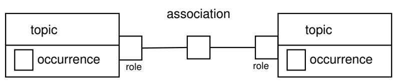


Figure 1: The basic constructs of a topic map. *Occurrences* are optional. Note that an association *must* consist of at least two members. Players of the association are topics. Topics and associations can be reified and are topics again.

steps to complete the system where each iteration step refines the model. The final model in this refinement chain is concrete enough so that generators can be used to create executable code. Ideally, the developer never sees source code and operates at a higher level of abstraction, leading to increased efficiency. In brief, the MDA motto can be summarized as "the model is the code". In XP, the process is reversed and the code is used as a model: XP development concentrates on source code maintenance and tries to make the code as expressive as possible, e. g. by choosing "self-explaining" variable and method names. Code-centricity has the advantage of allowing the developer to quickly react to changing requirements without having to constantly update design artifacts (as is done in traditional development). But both methods have their disadvantages: MDA (as executable UML) cannot yet be used for general-purpose programming. Additionally, a lot of complexity is hidden in generators. When having to adapt them, one is back to writing regular source code, in a manner that is much more difficult than regular coding. In XP, the expressiveness of normal source code is rarely enough for clearly displaying all of the modeling knowledge. This leads us to our tool called TUNA (TUBE Navigator). It combines the strengths of both processes: the abstraction abilities of MDA and the generality of XP. To that end, we created a generic ontology for source code. This ontology is instantiated as *topic maps*, a standard for knowledge representation. Topic maps are well suited for the task of integrated source code management, for the following reasons [11]: they are able to integrate a variety of artifacts in addition to source code; every construct is explicit and can be annotated very flexibly (even annotations themselves, recursively); annotations do not interfere with applications that are unaware of them; topic maps are based on very few basic constructs which makes querying very versatile. TUNA is geared towards moving XP in the direction of MDA by giving it richer means for expressing the intention of source code. Additionally, one gains powerful new ways of exploring the structure of source code.

This paper is structured as follows: We define topic maps and give an ontology for code. Then we present our code browsing tool and show how annotation is currently handled. We conclude by mentioning related work and future research.

2 Topic Maps

Topic Maps [3] are an ISO standard for graph-based knowledge representation. The three basic constructs in a topic map are: *Topics*, *associations*, *occurrences* and *roles* (Fig. 1). Topics are the nodes in the graph and can contain fields called *occurrences* which are just a set of key-value pairs. Occurrences provide

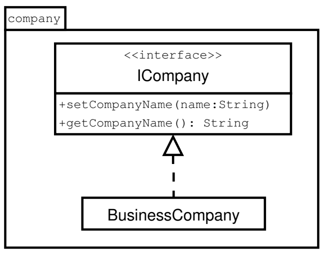


Figure 2: The UML diagram for the example package `company`. It contains an interface `ICompany` and its implementation `BusinessCompany`.

contextual information about a topic such as naming the resources it refers to. For example, a topic about a person could contain her/his email address. Topics are connected by named edges called *associations*. An association can connect more than two topics. The topics connected by an association are called its members. Members are ordered by giving them *roles*. These are basically labels that add a sense of direction to the association. Part of the power of topic maps is that most constructs that are not topics can be *reified* as topics (turned into a topic so that it can be described and annotated with topic map means, inside the same topic map): Associations can be typed by a topic and reified as a topic, the key of an occurrence is a topic, as are the role labels.

3 Running Example

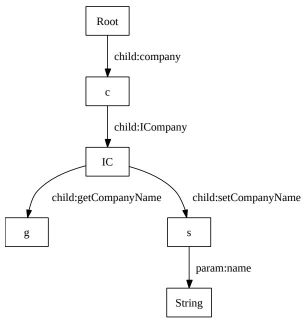


Figure 3: The structure of the example expressed as a topic map. We show a simplified version of the associations (i. e., there are no roles, just labeled arrows).

Our running example is a package that provides an interface for companies and an implementation of this interface (Fig. 2). This example is graphically expressed as a pseudo topic map in Fig. 3. The following excerpt in *linear topic map notation* (LTM) shows how it is represented as a real topic map. Unfortunately, explaining LTM is beyond the scope of this paper, please refer to [4] for further information. Hopefully it will give you at least an idea of what the topic map looks like. We only list on identifier and don't show modifiers and codeData occurrences.

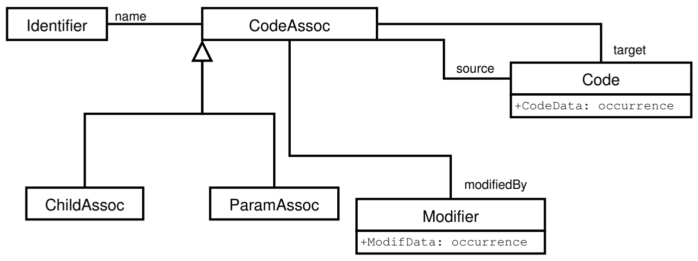


Figure 4: The Ontology for Tuna. We used the UML notation.

```
/* code topics */
[Root : Code = "Root"] /* empty root node */
[c : Code = "c"] /* package */
[IC : Code = "IC"] /* interface */
[g : Code = "g"] /* method */
[s : Code = "s"] /* method */
[String : Code = "String"] /* parameter */

/* associations */
childAssoc(Root: source, c: target, idCompany: name)
childAssoc(c: source, IC: target, idICompany: name)
childAssoc(IC: source, g: target, idGetCompanyName: name)
childAssoc(IC: source, s: target, idSetCompanyName: name)
paramAssoc(s: source, String: target, idName: name)

/* identifiers */
[idCompany : Identifier = "idCompany"]
{ idCompany, identData, [[Company]] } /* occurrence */
```

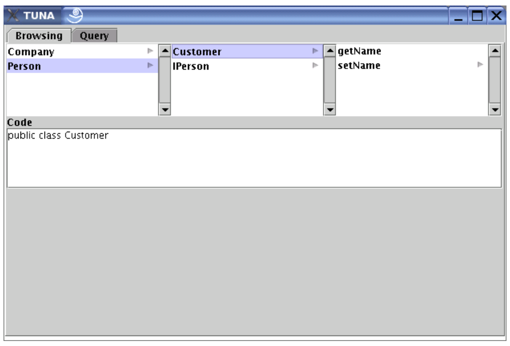
4 An Ontology for Code

So far, we have presented instances of our code ontology. The ontology that Tuna is based on is defined in UML Notation in Fig. 4. The ontology comprises the following concepts:

- **Code:** a code topic is a node containing source code as string or alternatively a reference pointer in URL form to an external ressource. In Java, classes and interfaces are code topics, and so are methods and their parameters). The relation between them can be modelled through the CodeAssoc-node.

- **CodeAssoc:** a CodeAssoc links two code topics. The CodeAssoc is of an given kind to express the relation between the two code topics. Further an identifier for the CodeAssoc is provided.

- **Kinds of code associations:** for now we only have two kinds of CodeAssocs, child and parameter. The child kind expresses a parent-child relation. The parameter kind is for code topics that are methods with parameters. Other kind of edges could easily be introduced.

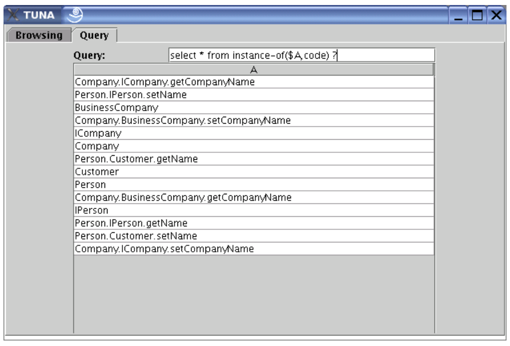


Company	Customer	getName
Person	IPerson	setName

Code

```
public class Customer
```

Figure 5: The browser tab displays the code topic map as a hierarchy (in the browser control in the top half of the window). The content of the currently selected topic is shown in the text field underneath.



Query: select * from instance-of($A,code) ?

A
Company.ICompany.getCompanyName
Person.IPerson.setName
BusinessCompany
Company.BusinessCompany.setCompanyName
ICompany
Company
Person.Customer.getName
Customer
Person
Company.BusinessCompany.getCompanyName
IPerson
Person.IPerson.getName
Person.Customer.setName
Company.ICompany.setCompanyName

Figure 6: The query tab consists of a text field for entering a Tolog query and a table underneath that displays the query results. It contains one column for every free variable used in the query.

- **Identifier:** A identifier is just a container for an identifier string for a CodeAssoc.
- **ModifierAssoc:** the modifier association deals with the relation of Code-Assocs and their modifiers. In Java these modifiers can be *public*, *private*, *protected*, etc. The modifier-association uses a reified edge topic for providing the additional information about the modifier(s).Further it just contains a simple string with the text for a certain modifier.

5 A Code Browsing Tool

The basic implementation was done in Java using the TM4J ("Topic Maps for Java") engine [8]. The API allowes us to create TopicMaps manually, to manipulate them in memory and to serialize them to a file. Query facilities are provided by the Tolog query language (a Prolog-like language, see [5] and [1]) that makes it easy to perform queries in an SQL-like manner. The TUNA window has two tabs: one for browsing a topic map (Fig. 5) and a second one for expressing queries in Tolog (Fig. 6).

6 Annotations

Currently, custom annotations still have to be added externally. As a quick demonstration, we add one to our running example. Let's say we want to attach a textual note to the interface `ICompany`. As before, we use the Linear Topic Map Notation and add the lines listed below to the text file that contains the topic map with the running example. First, we introduce three topics: `Note` for typing the annotation, `noteContent` for the occurrence that will contain the content of the note and `annotation` for typing the association that connects the annotation to the code topic.

```
[Note        = "Note"]
[noteContent = "noteContent"]
[annotation  = "annotation"]
```

Then we create the note topic itself: In the first line, we specify a topic whose identifier is `ICompanyNote` and whose type is `Note` (where `Note` refers to the topic we have introduced before—the two topics will be connected by a typing association). The text string that is given last is a human readable name used for displaying the topic. The second and third line define the content of the note as an occurrence in `ICompanyNote`.

```
[ICompanyNote : Note = "ICompany Note"]
{ICompanyNote, noteContent,
    [[This is the ICompany interface. Further details: ...]] }
```

Next, we connect the note and the code topic. The roles are implicit here, the default is to use the type of the topic. Therefore, the first role is `Code`, the second role is `Note`.

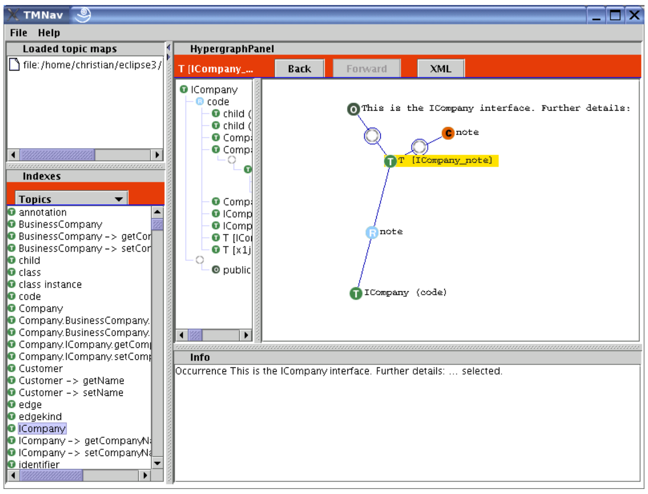
```
annotation (IC, ICompanyNote)
```

As a final test, we show the advantage of using a standardized data format: All tools that are based on that data format can be used. We display the annotated topic map in `TMNaV`, a generic topic map navigator based on TM4J (Fig. 7). The elegance of topic maps is that the annotated topic map can still be displayed and browsed in TUNA.

7 Related Work

JQuery [7] is a source code navigation and query tool integrated in the Eclipse IDE. All processing, navigating and searching is performed on a tree-based (and therefore hierarchical) data structure. This limits the modeling abilities of the data structure.

Another system which concentrates on the searching aspect is S4 [6]. S4 is a source code search and classification system that is based on meta-data and ontologies. It uses RDF (Resource Description Framework, [14]) for information storage, where we use topic maps. Further they focused on the classification aspect, while we are focusing on the modeling aspect. Still they are at a more granular level provided by three ontologies where we have only one. But querying a topic map provides much more power than the hierarchical system of S4. What we can think of is defining any "view" at the system with the ability of dynamic resizing, while S4 is to static.



Occurrence This is the ICompany interface. Further details: ... selected.

Figure 7: The annotated topic map displayed in TMNav. The highlighted topic in the center is the note, to its top right you see its type, to its top left the occurrence it contains. The bottom topic is the code that we have annotated, it is joined with the note by an association.

In Tuna we used an ontology as meta-model for modeling. In [13], a meta-model for refactoring is showed by the examples of Java and Smalltalk Source Code. They defined a meta-model and certain mappings in FAMIX for each single language like C/C++, Java, ADA etc. They have no real ontology but the mapping is similar. The starting point is a meta-model.

8 Future research

Future research for Tuna will concentrate on extending the tool so that it becomes a full-fledged environment for the ontology-based programming language Tube [12]. It will then include features such arbitrary annotations for code topics, meta-programming based on annotations etc. We are also exploring the idea of integrating refactoring into Tuna. Implementing refactoring can build on the formal foundation (i. e., the ontology) we have laid in this paper.

9 Conclusion

In this paper, we have presented a way of enhanced ontology-based software modeling. Topic maps provided a solid foundation for flexible storage, representation and retrieval of source code structure and for seamless integration of non-code artifacts.

References

[1] Ontopia A/S. tolog,Language tutorial. Web, 2004. `http://www.ontopia.net/omnigator/docs/query/tutorial.html`.

[2] K. Beck. *Extreme Programming Explained: Embrace Change.* Addison–Wesley, 1999.

[3] M. Biezunski, M. Bryan, and S. Newcomb. Iso/iec 13250, topic maps (second edition), 2002.

[4] L. Garshol. The Linear Topic Map Notation. Definition and introduction, version 1.2, 2002. `http://www.ontopia.net/download/ltm.html`.

[5] L. M. Garshol. tolog,A topic map query language. Web, 2002. `http://www.ontopia.net/topicmaps/materials/tolog.html`.

[6] K. Hosozawa and T. Ogihara. Source Code Search System Using The Knowledge Framework of The Semantic Web, 2004.

[7] D. Janzen and K. De Volder. Navigating and querying code without getting lost. In *Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 178–187. ACM Press, 2003.

[8] K. Ahmed et al. TM4J: a topicmap engine for Java, 2004. `www.tm4j.org`.

[9] J. Miller and J. Muskerij. MDA Guide Version 1.0.1. OMG Document: omg/2003-06-01, 2003.

[10] S. Pepper and G. Moore. XML Topic Maps (XTM) 1.0, 2001. `http://www.topicmaps.org/xtm/`.

[11] A. Rauschmayer and P. Renner. Knowledge-Representation-Based Software Engineering. Technical Report 0407, Ludwig-Maximilians-Universität München, Institut für Informatik, May 2004.

[12] A. Rauschmayer and P. Renner. Tube: Interactive Model-Integrated Object-Oriented Programming. Submitted for publication, 2004.

[13] S. Tichelaar, S. Ducasse, S. Demeyer, and O. Nierstrasz. A meta-model for language-independent refactoring. In *Proceedings ISPSE 2000*, pages 157–167. IEEE, 2000.

[14] W3C. Resource Description Framework (RDF), 2004. `http://www.w3.org/RDF/`.