TCD: A Text-Based UML Class Diagram Notation and Its Model Converters

Hironori Washizaki, Masayoshi Akimoto, Atsushi Hasebe,
Atsuto Kubo, and Yoshiaki Fukazawa

Department of Computer Science and Engineering,
School of Fundamental Science and Engineering, Waseda University,
3-4-1 Okubo, Shinjuku-ku, Tokyo 1698555, Japan
washizaki@waseda.jp,
{aki12,a-hasebe,a.kubo}@fuka.info.waseda.ac.jp,
fukazawa@waseda.jp
http://www.washi.cs.waseda.ac.jp
http://www.fuka.info.waseda.ac.jp

Abstract. Among several diagrams defined in UML, the class diagram is particularly useful through entire software development process, from early domain analysis stages to later maintenance stages. However conventional UML environments are often inappropriate for collaborative modeling in physically remote locations, such as exchanging models on a public mailing list via email. To overcome this issue, we propose a new diagram notation, called "TCD" (Text-based uml Class Diagram), for describing UML class diagrams using ASCII text. Since text files can be easily created, modified and exchanged in anywhere by any computing platforms, TCD facilitates the collaborative modeling with a number of unspecified people. Moreover, we implemented model converters for converting in both directions between UML class diagrams described in the XMI form and those in the TCD form. By using the converters, the reusability of models can be significantly improved because many of UML modeling tools support the XMI for importing and exporting modeling data.

1 Introduction

Unified Modeling Language (UML[1]) is a standardized diagram-based modeling language in the field of software/system engineering, especially object-oriented software development. Among several diagram specifications defined in UML, the class diagram is particularly useful through entire development process, from early domain analysis stages to later deployment/maintenance stages. UML class diagrams are used for expressing the static structure of some targets, such as problem domains, systems, and software, by describing the internal structure (attributes and operations) of classes and the interrelationships (e.g. generalization and association) between each class.

T.-h. Kim et al. (Eds.): ASEA 2010, CCIS 117, pp. 297–303, 2010.

The main description methods used for UML class diagrams are descriptions made using UML modeling tools and handwritten notes. Modeling tools such as `astah*`[2] and Rational Rose[3] enable intuitive and advanced model editing using GUI, and provide advanced functions such as validation of descriptions and source code generation. On the other hand, handwriting is a simple method of description regardless of the situation.

However, there are some situations in which the above two description methods are difficult to use. One such situation is collaborative modeling with a number of unspecified people in physically remote locations, such as exchanging models on a public mailing list via email. Figure 1 shows an example of an actual class diagram[1] presented at a mailing list[6]. This class diagram is expressed using characters like "+" and "-," and discussions are held by attaching or describing class diagrams in e-mail messages; however this diagram has been described in an ad-hoc manner.

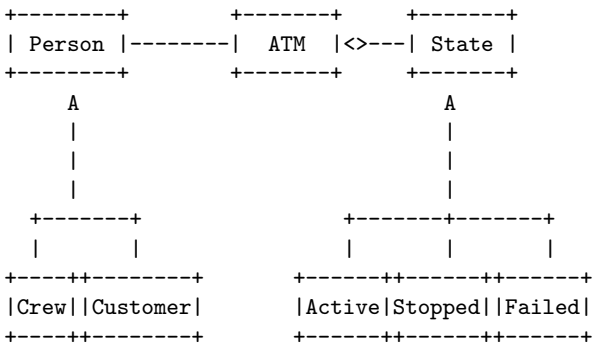

Fig. 1. Actual class diagram taken from a discussion mailing list[6]

To describe, share, edit and reuse class diagrams in the discussions held on mailing lists, a common format with a clear grammar is necessary for exchanging class diagrams to enable those to be interpreted exactly in the same way under any reader's environment. Although most of modeling tools can output images in common formats such as JPEG, the images cannot be imported or edited by another user's modeling tool.

Many modeling tools support the XML Metadata Interchange (XMI[4]) as a common specification based on text format for saving data. XMI is a standard specification for the exchange of models conforming to the MetaObject Facility specification (MOF[5]) in the form of XML documents. XMI data is described in the form of text, but it is described in a way that allows it to be understood easily by computers (i.e. machine-readable); it is not easy for humans to understand text in XMI (i.e. NOT human-readable), so users who do not have modeling tools cannot utilize such modeling data. Thus, XMI is not well suited to work environments where many people need to view and modify the data. Models

[1] Texts in the diagram are originally described in Japanese.

described in handwritten form are also inappropriate for collaborative modeling because it is difficult to deliver and share models in this form.

To overcome the above-mentioned problems in conventional description methods, formats and specifications, we propose a new diagram notation, called "TCD," for expressing UML class diagrams using ASCII text format. Moreover, we implemented model converters for converting in both directions between UML class diagrams described in the XMI form and those in the TCD form.

The remainder of this paper is organized as follows. Next section introduces the concept and feature of TCD. Section 3 describes our model converters. Section 4 describes several related works. In the last section, we draw a conclusion and state future works.

2 Text-Based UML Class Diagram

We defined TCD based on the following concepts to make it truly useful for above-mentioned collaborative modeling situations.

- Conformity to UML class diagram specification: TCD conforms to most of features defined in UML class diagram by using ASCII text format. In TCD, each class definition is described as an independent text element. The specification of the class description is defined as an extension of a conventional text-based notation called "U–Language"[7]; we have added several important features that are not supported in U–Language such as `static`/`final` members of a class, and an `abstract` class.
- Balancing machine-readable and human-readable: TCD supports the description of associations among classes by using mainly two characters: "|" and "-". Thus TCD can handle not only horizontal lines in association definitions, but also vertical lines, which enables association definitions to be description in a way that makes overall structure easy to understand.
- Built-in conversion between TCD and XMI formats: we developed tools for TCD to XMI data conversion, and for XMI to TCD conversion, to enable the migration between TCD and modeling tools. By using the tools, users can continue modeling activities started in a different format.

For example, the UML class diagram of the Abstract Factory design pattern[10] shown in Figure 2 can be described by the combination of the class definitions in the left side of Figure 3 and the association definitions in the right side of Figure 3 in TCD.

By using the example, details of TCD descriptions are explained in below.

(1) Class Definition

The class definitions describe the details of the classes that appear in the class diagram. For example, the left side of Figure 3 shows definitions of two classes: `AbstractFactory` and `ConcreteFactory1` taken from the Abstract Factory

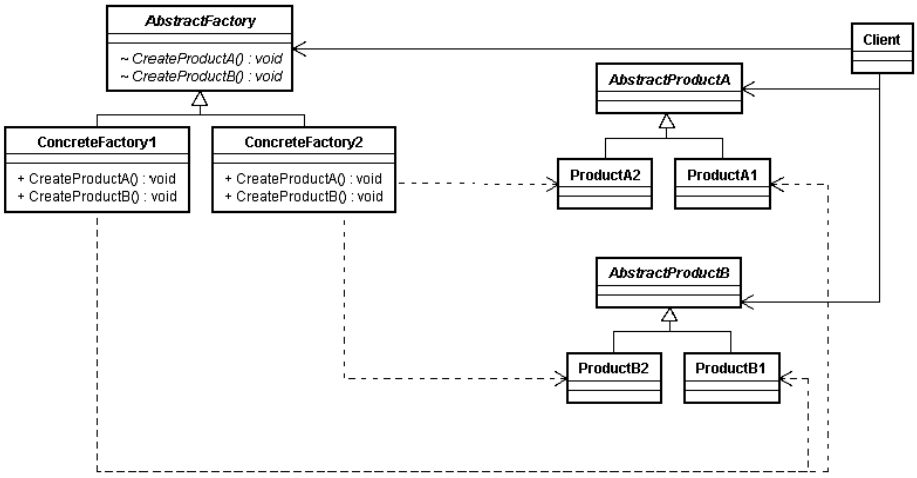

Fig. 2. UML Class diagram describing the AbstractFactory pattern

pattern[10]. In the figure, two methods with precise signature definitions are specified for each class.

As shown in Figure 3, each class is specified by writing it between upper and lower boundary lines formed using "`=`" characters. Furthermore, two lines formed using "`-`" characters are inserted between the boundary lines, to create three compartments: class name, attributes and methods.

Since `AbstractFactory` is an abstract class, "`&`" is written before the class name. And since it does not have any attributes, nothing is written in the second compartment from the top. In the lowest compartment, two methods are specified. To declare the method, "`()`" is added after the method name. Return value type is specified by writing "`:`" after the method name and writing the type name.

(2) Association Definition

Associations are expressed by specifying the two classes that are associated with each other, and drawing lines with arrows to indicate the type of each association between the classes.

For example, the right side of Figure 3 shows 10 classes and 13 associations. Among them, the line emerging from the lower side of `AbstractFactory` is described using the characters "`^`", "`-`", and "`|`", indicating that the class generalizes the classes listed at the opposite end of the line; the next character is "`+`", and here the generalization line emerging from `AbstractFactory` branches. "`*`" and "`#`" indicate the association's rotation point and intersection point, respectively.

```
// Class definitions                // Association definitions


==============================      Client--->AbstractFactory
    & AbstractFactory               |             ^
------------------------------      |             -
------------------------------      |             |
 + CreateProductA() : void          |         *---+--------------*
 + CreateProductB() : void          |         |                  |
==============================      |       ConcreteFactory1   ConcreteFactory2
                                    |            ^                     ^
                                    |            :                     :
==============================      |            :                     :
    ConcreteFactory1                |         *..+..............*     :
------------------------------      |         :                   :    :
------------------------------      |         :          *........#....+......*
 + CreateProductA() : void          |         :          :        :           :
 + CreateProductB() : void          |  ProductA1  ProductA2    ProductB1  ProductB2
==============================      |         |          |        |           |
                                    |         |          |        |           |
                                    |         -          -        -           -
==============================      |         v          v        v           v
    ConcreteFactory2                |       AbstractProductA   AbstractProductB
------------------------------      |         ^                  ^
------------------------------      |         |                  |
 + CreateProductA() : void          *---------+------------------*
 + CreateProductB() : void
==============================
```

Fig. 3. TCD description of the AbstractFactory pattern (left side: classes, right side: associations)

Moreover, users can describe associations by simply listing up each association such as `AbstractFactory <--- Client` and `AbstractFactory <|--- ConcreteFactory1`. TCD enables users to select whether to emphasize ease of description (like the above-mentioned listing up) or ease of understanding (like Figure 3).

Multiplicities and roles (association ends) regarding associations can be specified for those described horizontally by using brackets ("`( multiplicity )`") and square brackets ("`[ role ]`") for each end. For example, `ClassA (1) ---> (*) ClassB` denotes one-to-many relationship from `ClassA` to `ClassB`.

3 Model Converters

We developed tools for TCD to XMI data conversion, and for XMI to TCD conversion, to enable the migration between TCD and modeling tools. By using the converters, users can continue modeling activities started in a different format.

We implemented converters by using Java and JavaCC[11], shown in Figure 4. Since these converters can be used on any system that can run Java programs, it can be used very widely. Figure 2 shows an UML class diagram by inputting

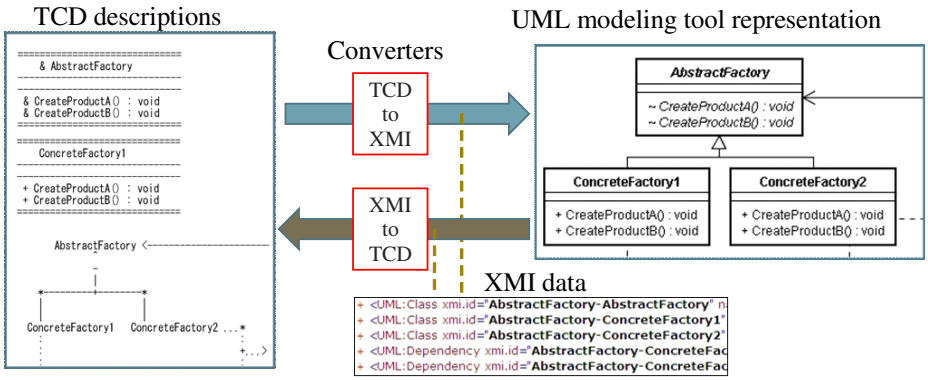

Fig. 4. TCD converters and related formats

TCD descriptions in Figure 3 into the converter, and importing the XMI output of the converter into a modeling tool `astah*`.

It is found that all of contents described in TCD are kept in standard UML notation. The converters allow for easy collaboration between users of TCD and modeling tools. For example, these converters allow work on a model that uses class diagrams initially written in TCD to be continued using a modeling tool. They also enable conversion of class diagrams created by a modeling tool into TCD format for exchange in email-based discussions.

4 Related Work

Although not in mainstream use, there are several methods for describing UML class diagrams based on text, such as U–Language[7] and Silvertejp[9]. These methods enable user write class diagrams into text forms such as e-mail and Wikis, where conventional description methods are difficult to apply.

However, existing description formats are equipped with on-way converters to enable reutilization of described models; these converters are only capable of converting from each text description format to other formats, or from other formats to the text description form. For example, there is a converter that outputs Java source code from descriptions in U–Language[8]; however there is no support for converting Java source code into U–Language descriptions. Furthermore, there is no compatibility between different description formats and it is also difficult to make use of other converters, so the reusability of models is low.

In addition, existing formats for text description-based class diagrams tend to feature either very good ease of description or very good ease of understanding (not both); it is hard to use one format adaptively for different situations such as a case in which ease of description is most important or another case in which ease of understanding is most important.

5 Conclusion and Future Work

We formulated a new kind of text description-based method to express UML class diagrams – Text-based Class Diagram (TCD) – to overcome the problems with conventional description methods, and we developed converters to convert between TCD and XMI formats.

TCD conforms in part to the description specifications of the existing description formats that offer good ease of description, and enables the expression of vertical associations, which allows for layouts that are easy to understand. These features enable users to select between description that emphasizes ease of description and description that emphasizes ease of understanding, according to the application. In addition, the converters make collaboration with users of modeling tools easy and highly reliable, thereby improving model reusability.

As our future work, we have a plan to conduct real experiments for confirming the usefulness of TCD and its converters compared with conventional environments. Moreover, since many large-scale class diagrams make use of package-related notation, it is necessary to support package-related notation in order to expand the range of class diagrams that can be handled using TCD.

Acknowledgements

This research has been partially supported by the Mizuho Foundation for the Promotion of Sciences (2009-2010) and the JGC-S SCHOLARSHIP FOUNDATION (2010-2011).

References

1. Object Management Group: Unified Modeling Language (UML), `http://www.uml.org`
2. Change Vision, Inc.: astah* - UML and Mind Mapping Integrated Modeling Tool, `http://astah.change-vision.com/en/`
3. IBM: Rational Rose, `http://www-306.ibm.com/software/awdtools/developer/rose/`
4. Object Management Group: MOF 2.0/XMI Mapping Specification, `http://www.omg.org/technology/documents/formal/xmi.htm`
5. Object Management Group: MetaObject Facility, `http://www.omg.org/mof/`
6. Ogis-RI: Object Square, `http://www.ogis-ri.co.jp/otc/otc2/oosquare-ml/`
7. Hiranabe, K.: U–Language – Human and machine readable UML text format, ObjectClub (in Japanese), `http://www.objectclub.jp/technicaldoc/uml/u_lang/`
8. Hexagonta: U Language Parser, `http://sourceforge.jp/projects/ulparser/`
9. Wettin, K.: Silvertejp, `http://silvertejp.tigris.org/`
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.M.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1994)
11. Viswanadha, S.: Java Compiler Compiler (JavaCC), `https://javacc.dev.java.net/`