Portable Mapping of Uml Design Specifications into Object Oriented Code

Swati Nair
K.K.Wagh Institute of Enginnering Education & Research, Nashik, MH, India.

Hemangini Mahajan
K.K.Wagh Institute of Enginnering Education & Research, Nashik, MH, India.

Mayur Sangtani
K.K.Wagh Institute of Enginnering Education & Research, Nashik, MH, India.

Sharad Jondhale
K.K.Wagh Institute of Enginnering Education & Research, Nashik, MH, India.

Anisha Lamba
K.K.Wagh Institute of Enginnering Education & Research, Nashik, MH, India.

ABSTRACT

This paper proposes a stand-alone software that accepts UML Design Specifications through a simple user interface. It will generate XML code and object-oriented code for the specifications provided by the user, in parallel. Along with the generation of code, the software will also produce the graphical representation of the UML diagram, thus facilitating the easy visualization of the diagram being produced. This tool will free the software developer from the mundane task of writing simple class skeleton so he/she can concentrate on the business logic and overall architecture of his/her project, provides consistency between design and code and eliminates the unintended errors that can creep into manually written code. It will create an effective integration between the design and implementation stages. Also, due to generation of XML code, it facilitates portability. This paper concentrates mainly on class diagram and Java code that can be extended in future to cover other diagrams and other object-oriented languages**.**

General Terms

Object-oriented Modeling and design, Software Engineering, Object-Oriented Programming, Unified Modeling Language.

Keywords

Class diagram, GUI, object-oriented code, portable, UML design specifications, XML.

1. INTRODUCTION

In order to better design and manage the development of complex software systems, software designers have turned increasingly to modeling languages such as the Unified Modeling Language (UML). Relationships between the components of a system are grasped more easily when the design is represented graphically using a modeling language. UML modeling tools generally support the generation of skeletal implementation code either directly or by exporting models in a standardized format, such as XML, that can be used by third-party tools. Tools for the generation of code from model descriptions are valuable in helping developers maintain consistency between a model and its implementation, which may involve a large number of source files compared to size of the model.

In this paper, we study the creation of an independent software that will accept UML Design Specifications through a simple user interface and will generate XML code and object-oriented code for the specifications provided by the user, in parallel. Along with the generation of code, the software will also produce the graphical representation of the UML diagram, thus facilitating the easy visualization of the diagram being produced.

Our goal in the generation of skeletal implementation code and XML code from design specifications is to facilitate the design and development process by:

- Allowing designers the flexibility to model systems with few assumptions about the underlying implementation.
- Providing designers with a high-level, consistent framework based on the design that supports the use and extension of the system without exposing the implementation of its components.
- Providing implementers with a framework in which the implementation of each component of the design can be carried out independently of the implementation of the other components.
- Minimizing designer and implementer effort.
- Allow portability of class diagram by generating XML code.

2. DOMAIN ANALYSIS

This part covers the basic concepts that are considered in this software development.

2.1 Unified Modeling Language

The Unified Modeling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development. UML aims to be a standard modeling language which can model concurrent and distributed systems. UML is a de facto industry standard, and is evolving under the auspices of the Object Management Group (OMG).

2.2 Class Diagram

A class diagram is a structural diagram that lays out the object classes of a system, their properties and relationships. For the purpose of this project, we focus only on the class diagrams. The class diagram is the main building block of object oriented modeling. The purpose of the class diagram is to show the static structure of the system being modeled.

2.3 Extensible Markup Language

Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. XML is a new information format which allows data portability – the ability for different computer systems to exchange data without the need for specially written file translation programmes or "interfaces."

2.4 Object-Oriented Code

Object-oriented programming (OOP) is a programming paradigm using "objects" – data structures consisting of data field and methods together with their interactions – to design applications and computer programs. Clearer code makes maintenance, and future improvements, easier. It improves reusability.

3. FUNCTIONAL REQUIREMENTS

- Provision of simple user interface for accepting the UML design specifications
- Fully automated generation of XML code from given specifications
- Fully automated generation of OO code(Java).
- Simultaneous graphical representation of class diagram(tree structure)
- Error handling by displaying errors such as classname conflict i.e. the specifications should not match any keywords, etc.

4. SYSTEM ARCHITECTURE

We describe a novel approach of portable mapping of UML design specifications into object-oriented code. The input to the system is various class diagram specifications such as class names, attributes, operations and relationships.

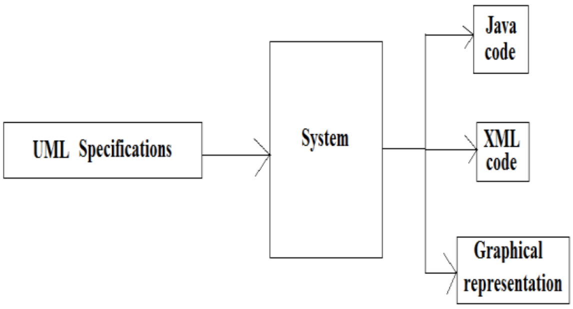


Fig 1: System Architecture

The system will consist of three modules: Mapping of specifications into : a) Java code , b) XML code, c) Tree structure of class diagram. These three functions will run in parallel. The transformation rules will be applied to convert the specifications in textual form to code. An instance of rule can be : generalization → inheritance. The main focus must be on implementation of GUI i.e. front end of the system. The back-end consist of two databases : one for storing reserve words present in java and the other for storing classes, attributes, methods and relationships like generalization, etc.

5. CONVERSION OF SPECIFICATIONS INTO OO CODE

This section outlines conversion of class diagram specifications into java class skeleton. So, we will concentrate on some elements of class diagram.

UML Class: Mapped to Java class. If the UML class is abstract, then add abstract keyword in code. We can also generate the class constructors for the classes.

UML attributes: It consists of attributes and association. A simple way of implementing attributes in code is with private data fields. Objects of the class can then retrieve or alter this data with public getter and setter methods. Multi-valued attributes are coded with a collection.

Operations**:** Operations are the actions of a class, and correspond to methods in Java.

UML associations: The classes involved in association will have an instance variable of another class.

UML dependency: Dependency is mapped to parameter, return value or local variable in operations of the class.

6. CONVERSION OF SPECIFICATIONS INTO XML

This section outlines conversion of class diagram specifications into XML schema. The main goal here is to describe the mapping for understanding how the actual conversion takes place in short. So, we will concentrate on some elements of class diagram. We follow a tabular format to represent this formalization to identify the UML entity being mapped, its equivalent XML schema notation and extensions (special case of mapping which slightly modifies the techniques of mapping but does not violate the general rule).

For example, Table I represents mapping specification for UML classes. We choose to represent base classes as abstract, only if they are represented by the stereotype <<abstract>> in the UML Model.

Table I.
Mapping Specification for UML Class

UML entity to be mapped	Class
Mapped XML Entity	XML element (<xs:element>) and a matching Complex Type (<xs:complexType>)declaration
Extensions	If a class has an <<abstract>> stereotype associated with it, or is italicized, it's abstract attribute is set to true.

7. EXPERIMENTARY RESULTS

This section covers the User Interface of the proposed tool and the sample code that will be generated. Fig. 2 shows the class specification panel in which we will accept the class name, package name, stereotype, access specifier, base class and interface. Fig. 3 shows the attributes panel and Fig. 4 shows the methods panel in which we will accept the attributes and methods of the class respectively. Also we will have a panel for accepting the relationships of the class with other classes. Fig. 5 shows the main window alongwith the sample XML and Java code generated. Also, sample tree structure is shown.

8. CONCLUSION

This project work aims to bridge the gap between design and implementation phases. We consider the domain of XML and OO code and provide a method to map UML specifications into XML and Java code. A few diverse approaches have been suggested towards forward engineering. Currently, Rational Rose has a feature to convert UML diagrams into code, but the code is generated after creation of UML diagrams. Other tools available are : MyEclipse UML, ArgoUML, Altova, etc. Most of them are web based tools and users must have good knowledge about the respective tools. The available systems generate the code from diagrams. The users have to drag and drop the notations as well as enter specifications to create diagram and then the code is generated. This software will allow users to create code by only entering UML specifications. This software is needed in time-critical projects since this tool accelerates the implementation phase of the project. It is needed to software developers so that he/she can concentrate on the core logic of his/her project and will free him/her from the task of writing simple class skeleton. It is also needed for portability of diagrams.

9. ACKNOWLEDGMENTS

We would like to thank our project supervisor, Mrs. Snehal Kamalapur for highlighting the idea of “Portable Mapping of UML Design Specifications into Object Oriented Code”. We would also like to thank Mr. Mohit Makhija of Persistent Systems Ltd. for his support for our sponsored project. We gladly take this opportunity to thank Prof. Dr. K. N. Nandurkar, Principal, KKWIEER and Prof. Dr. S. S. Sane, Head of Dept., Computer Engineering for providing facilities during progress of the project. We are thankful to all those who helped us directly or indirectly to develop this project report.

10. REFERENCES

[1] Booch, G., Rumbaugh, J. and Jacobson, “The Unified Modeling Language User Guide”, Addison- Wesley, Reading, MA, 1999.

[2] University of California, “ArgoUML: An Object Oriented UML Design Tool”, Tigris Open Source 2001.

[3] Justin Elsberry and Nicholas Elsberry, “Using XML and SVG to Generate Dynamic UML Diagrams” .

[4] W3C Architecture domain:Extensible Markup Language (XML). W3C, 2003.

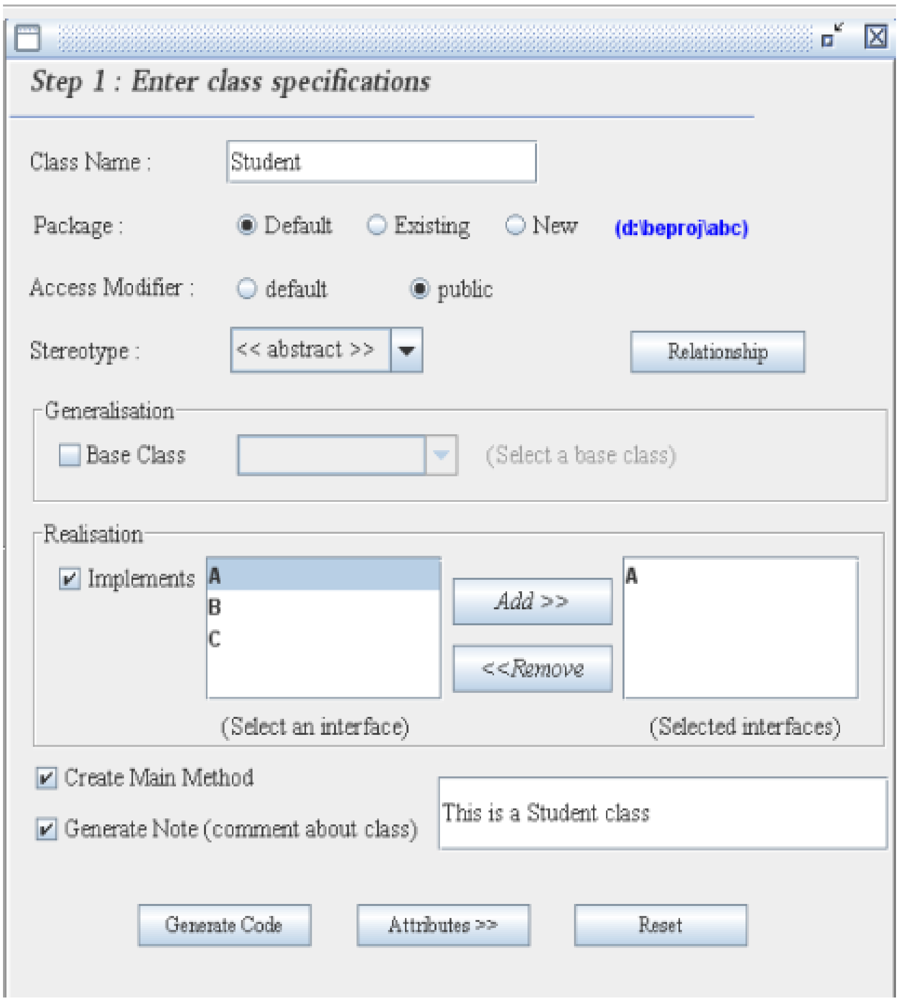
[5] XML in 10 points. W3C, 2001.

[6] William Harrison, Charles Barton, Mukund Raghavachari, “Mapping UML design to Java”.

[7] Krish Narayanan, Shreya Ramaswamy, “Specifications for Mapping UML Models to XML Schemas”.

[8] Robert C. Martin, ”UML Tutorial- Class diagram”.

[9] Robert C. Martin, ”UML for Java Programmers”.



Step 1 : Enter class specifications

Class Name : Student

Package : ● Default ○ Existing ○ New (d:\beproj\abc)

Access Modifier : ○ default ● public

Stereotype : << abstract >>

Generalisation

□ Base Class (Select a base class)

Realisation

☑ Implements

A
B
C

(Select an interface)

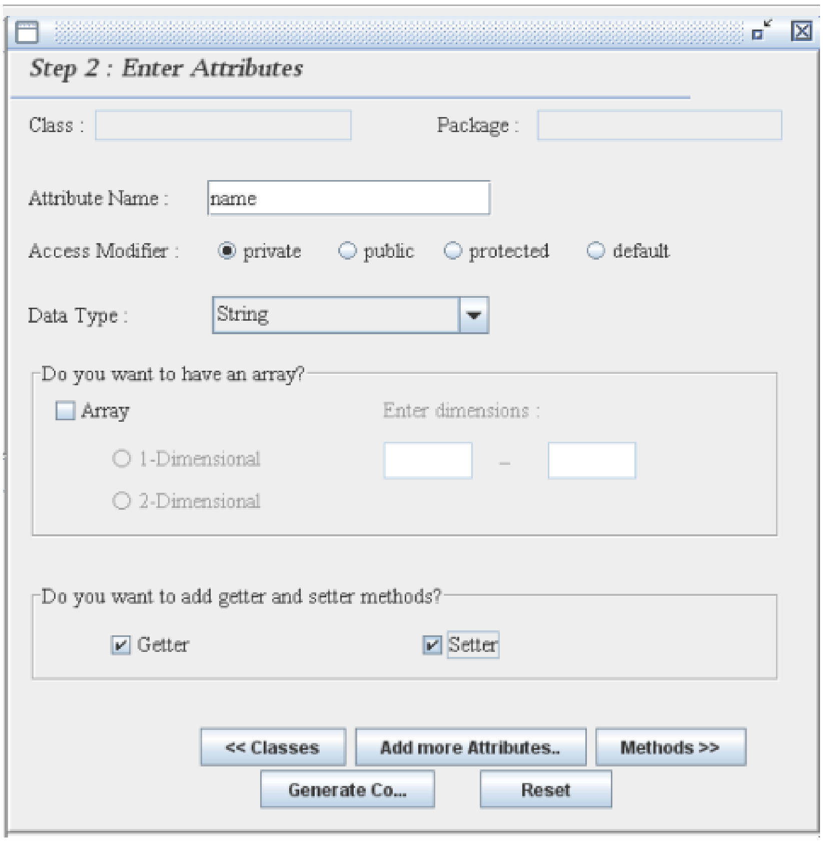
A

(Selected interfaces)

☑ Create Main Method

☑ Generate Note (comment about class) This is a Student class

Fig 2: Class specification Panel



Step 2 : Enter Attributes

Class :

Package :

Attribute Name : name

Access Modifier : ● private ○ public ○ protected ○ default

Data Type : String

Do you want to have an array?

□ Array

Enter dimensions : _

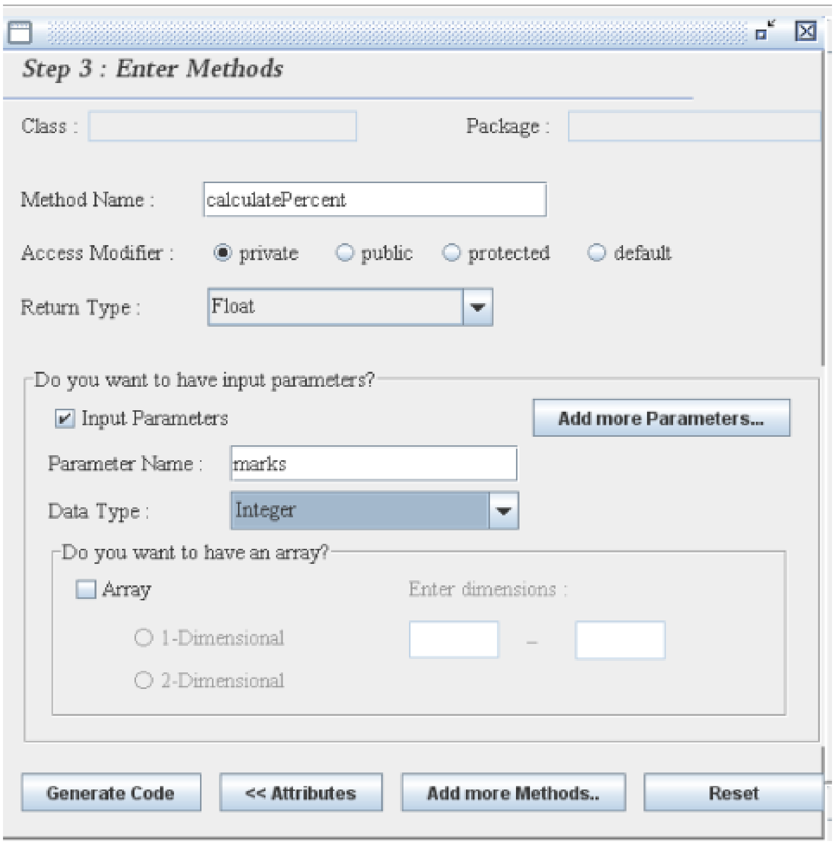
○ 1-Dimensional

○ 2-Dimensional

Do you want to add getter and setter methods?

☑ Getter ☑ Setter

Fig 3: Attributes Panel



Step 3 : Enter Methods

Class :

Package :

Method Name : calculatePercent

Access Modifier : ● private ○ public ○ protected ○ default

Return Type : Float

Do you want to have input parameters?

☑ Input Parameters

Parameter Name : marks

Data Type : Integer

Do you want to have an array?

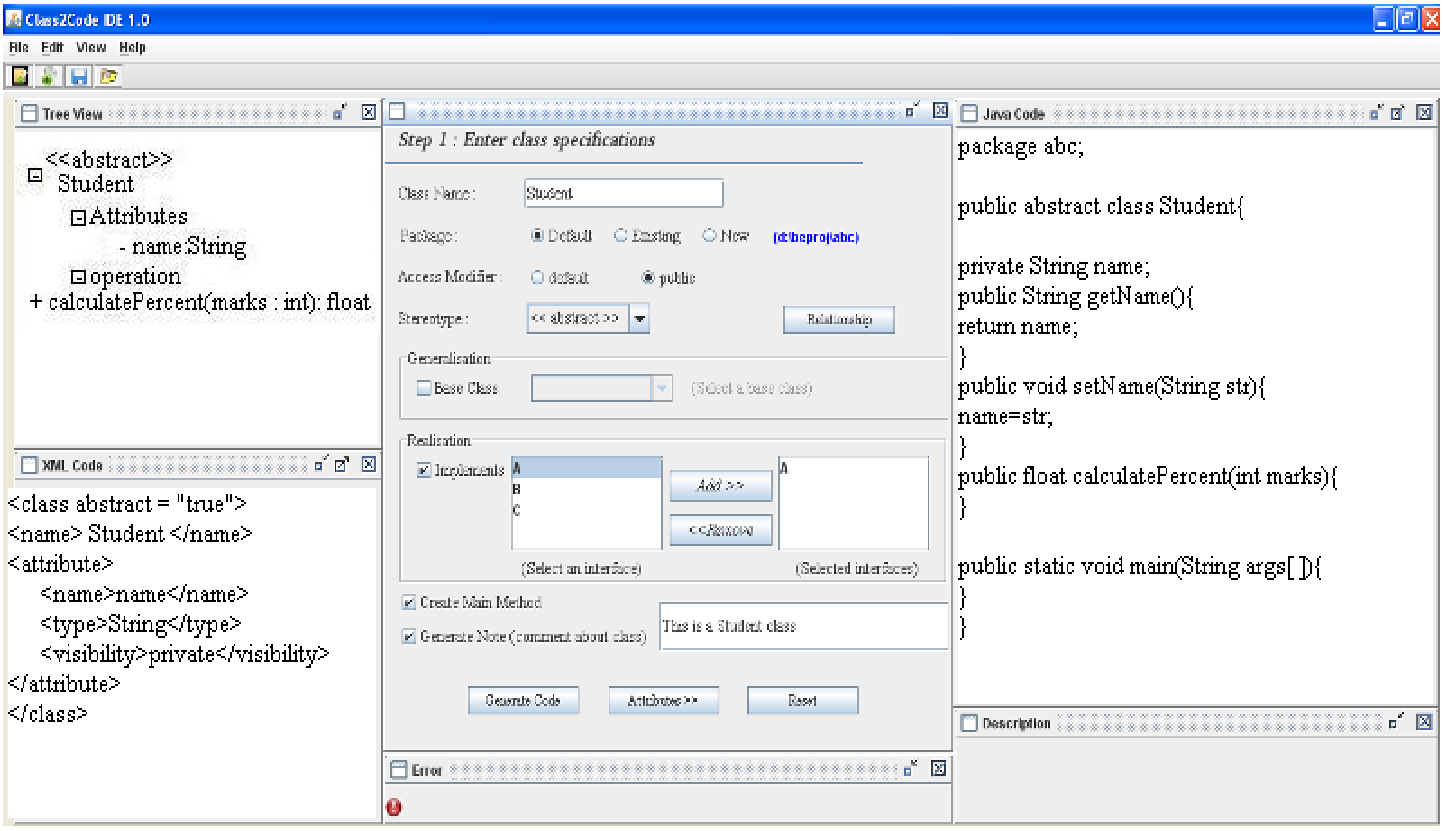
□ Array

Enter dimensions : _

○ 1-Dimensional

○ 2-Dimensional

Fig 4: Methods Panel



Tree View:

```
<<abstract>>
⊟ Student
    ⊟Attributes
        - name:String
    ⊟operation
+ calculatePercent(marks : int): float
```

XML Code:

```
<class abstract = "true">
<name> Student </name>
<attribute>
    <name>name</name>
    <type>String</type>
    <visibility>private</visibility>
</attribute>
</class>
```

Step 1 : Enter class specifications

Class Name : Student

Package : ● Default ○ Existing ○ New (d:\beproj\abc)

Access Modifier : ○ default ● public

Stereotype : << abstract >>

Generalisation
☐ Base Class (Select a base class)

Realisation
☑ Implements
A
B
C
(Select an interface)
A
(Selected interfaces)

☑ Create Main Method

☑ Generate Note (comment about class) This is a Student class

Java Code:

```
package abc;

public abstract class Student{

private String name;
public String getName(){
return name;
}
public void setName(String str){
name=str;
}
public float calculatePercent(int marks){
}

public static void main(String args[ ]){
}
}
```

Fig 5: Main window with sample code generation