A General-purpose Transformation System for Java

Gustavo Santos, Paulo Borba, Adeline Sousa
Informatics Center, Federal University of Pernambuco
{gas,phmb,adss}@cin.ufpe.br

Introduction

Program transformation can be considered as a unifying concept for code generation and refactoring. A refactoring comprises several behavior preserving changes on the program, but does not add new functionalities [3]. A code generation tool, on the other hand, introduces new functionalities. With this unifying view, transformations create new types and modify old ones as long as it preserves the semantics of the original program.

Most code generation and refactoring tools implement only a fixed set of transformations. This is quite restrictive because new refactorings and code patterns are often proposed, and some might be strongly related to specific design patterns, frameworks, and middleware.

In order to avoid this limitation, we are developing JaTS (Java Transformation System) a language and execution engine for defining and applying transformations. Using this language, users are able to define new transformations by not only composing existing ones but also by declaring preconditions and source and target templates that describe the code changes required by a transformation. This language's syntax is basically an extension of Java syntax with meta-programming constructs such as meta-variables, which are used as placeholders in templates. Besides meta-variables, the language provides more powerful meta-programming constructs [1,2] such as optional, conditional and iterative constructs. The language also has executable declarations that can have access to lower level code structures (syntactic trees elements) when necessary.

Language-specific versus general transformation tools

Many program transformation tools are not language-specific, being able to transform programs from an arbitrary encoded source language to an arbitrary destination language. Although this may be an advantage, it complicates the use of the tools, since the language in which the transformations are encoded is substantially different from the one to which they are applied.

There are also language-specific tools for program transformation. Most of these have the limitation of supporting only a fixed set of transformations. Some tools offer the possibility of extension through the use of an API, but this demands the user to access the system source code and actually implement the transformations using a programming language.

JaTS avoids the drawbacks of both general purpose and limited language-specific transformation tools. As JaTS's syntax is an extension of Java's syntax, it is easier for Java programmers to specify the transformations they wish to apply. Also, the transformation language takes the semantics of Java into account, for example adopting associative-commutative matching (for field and method declarations), which allows concise implementation of transformations that could be much more complicated to implement if only the syntax was taken into account.

Transformations in JaTS

JaTS transformations consist of three parts: preconditions, source and target templates. The templates consist of one or more type (class or interface) declarations. The type declarations in the source templates are matched with the source Java type declarations to be transformed; this implies that both must have similar syntactic structures. The target templates define the general structure of the types that will be produced by the transformation.

The following example shows a simple JaTS transformation that introduces a new field declaration in an arbitrary class. The construct “#Class_name” represents a typical JaTS variable. It will match with the real class name in matching process. The delimiters “#[” and “]#” indicate an optional matching. This means that the template shown in the example cans match successfully with a class declaring or not an extends clause. The variables “#Fds” and “#Mds” express the JaTS

semantic power. They can accomplish all field declarations and method declarations in the class respectively.

Source template

```
public class #Class_name
       #[ extends #Super_class ]#
{
    FieldDeclarationSet: #Fds;
    MethodDeclarationSet: #Mds;
}
```

Target template

```
public class #Class_name
       #[ extends #Super_class ]#
{
    private int newField;
    FieldDeclarationSet: #Fds;
    MethodDeclarationSet: #Mds;
}
```

Source class

```
public class ExampleClass
{
    private String oldField;
    public void method() {
    }
}
```

Transformed class

```
public class ExampleClass
{
    private int newField;
    private String oldField;
    public void method() {
    }
}
```

The application of JaTS transformations is basically based on matching, replacement and processing. The matching retrieves the information from the source Java types. There is also the possibility of passing parameters to the transformation when the matching process is not enough to get all needed information. The replacement transverses the parse-tree and replaces occurrences of variables by the values mapped to them, and the processing is responsible for the evaluation of executable and iterative meta-programming declarations.

Applicability

JaTS is being used as the transformation engine inside Coder [4], a wizard-based tool that can be used for generation and maintenance of Java programs. This experience has shown that JaTS can be used successfully for several pattern language generations, like a data collection architectural pattern for EJB, Struts-based presentation layer and others. The generated code is fully executable and requires few adjustments to be completely functional. This reduces significantly the software development cost for Coder users.

Our experience developing JaTS has shown that the system can be used in a bootstrapping way, such that the tool can support a product-line of language-specific transformation systems. Thus, it is possible to derive new transformation systems in a relative inexpensive way. Such technique is possible because JaTS transformation system model can be used for any language due to the concept of matching, replacement and processing and the adoption of the visitors design pattern.

Nowadays the JaTS transformation model includes the advantages of a language-specific transformation system, because it is easy for a programmer to learn the transformation language, and enables the user to obtain the advantages of general transformation systems, because it is inexpensive to derive a new transformation system for another language.

References

[1] Fernando Castor and Paulo Borba. A language for specifying Java transformations. 5th Brazilian Symposium on Programming Languages, pages 236-251. May 2001, Curitiba, Brazil. Also presented at the Dagsthul Seminar on Program Analysis for Object-Oriented Evolution. Dagsthul, February 2003.

[2] Marcelo d'Amorim, Clóvis Nogueira, Gustavo Santos, Adeline Souza and Paulo Borba. Integrating Code Generation and Refactoring. Workshop on Generative Programming, ECOOP'02. Málaga, June 2002.

[3] Martin Fowler et al. Refactoring: Improving the Design of Existing Code. Addison Wesley. November 1999.

[4] Qualiti Coder Tool. http://coder.qualiti.com.