27 *JTransform* A Tool for Source Code Analysis

Holger Eichelberger
Jürgen Wolff von Gudenberg
Institut für Informatik, Am Hubland, 97074 Würzburg, Germany
`{eichelberger,wolff}@informatik.uni-wuerzburg.de`

Abstract

Code transformation and analysis tools provide support for software engineering tasks such as style checking, testing, calculating software metrics as well as reverse- and re-engineering. In a reverse step information is picked from a source text in order to facilitate its comprehension, its visualization or the check of agreed standards concerning coding quality or layout.
In this short paper (see [3] for details) we describe *JTransform*, a general Java source code processing and transformation framework. It consists out of a Java parser generating a configurable parse tree and various visitors (transformers, tree evaluators) which produce different kind of outputs.

27.1 Architecture

The main design goals for *JTransform* have been extendibility by subclassing and plugins, easy configuration of applications and ease of use. Therefore we extensively used design patterns like visitor, factory methods, abstract factory, composite or strategy.

The tool is split into two configurable phases, the front end and the back end.

27.1.1 Front End

A JavaCUP generated Java parser builds the parse tree. The parser is configurable in order to generate parse tree nodes containing different amounts of information. Therefore the parse tree is generated by a factory which consists of a set of factory methods. As default the node's syntactic structure and its source code position is registered, type information can be added. During the static type resolution information from imported classes, not part of the source code files may be collected.

27.1.2 Back End

The parse tree is then traversed by at least one visitor to transform the tree itself performing syntactical checks or static type resolution, or to produce the application-specific output. Generally, different back ends need different information stored in the parse tree nodes. The collection of method calls and the detection of general dependencies between classes , such as a cast expression can optionally be performed while calculating the static type references.

27.2 Configuration

In order to infer additional information, usually the nodes are subclassed and a specialized visitor, which controls the information inference mechanism is implemented. In order to tell *JTransform* to use the new nodes, a specialized nodes factory has to be implemented. Interpretation and implementation of options is possible.

All back ends should extend one class that provides configuration methods for redirection of input and output. Multiple back ends may be chained.

As an example of dynamic configuration by components the source code checking application implements different source code checks, each as a single component.

Furthermore, an XML configuration or project file contains a section of global features which are applied to all files, and at least one section which describes local features applied to a set of files specified in that section.

27.3 Applications

27.3.1 Visualisation by UML Diagrams

A visitor transforms the program into UMLscript that is input for the diagram drawing framework Sugibib [4]

27.3.2 Programming Conventions

Source code formatting and naming conventions are checked whether they conform to commonly agreed standards [2, 7].

Various visitors performing static source code analysis are applied in order to highlight dangerous code that may be the source for tricky errors. Take the facts that the statement following an if or for is not a block, or that an iterator changes accidentally the underlying collection, as examples.

27.3.3 Forbidden Classes and Packages

In an educational environment or for security reasons some classes or packages may be forbidden. The Java access controller restricts the usage of library classes by defining a policy file.this mechanism is extended with *JTransform*. As an example, dynamic loading of classes and reflection can be forbidden, if user code is not allowed to make references to the `ClassLoader` and its subclasses, the class `Class` and all classes in `java.lang.reflect`. This rule is simple but very restrictive, often it is sufficient to declare certain methods of `ClassLoader` as forbidden in the user code.

27.3.4 Refactoring

A simple application of *JTransform* is to change class names, method names, parameter names, package names, move classes between packages and move entire packages and, of course, to correct all references to these classes and packages in the rest of the source code. As a further application, the changes and incompatibilities which result from deleting an attribute or a method or when changing the type of an attribute or parts of the signature of a method can be displayed; of course the changes can be executed but this usually requires further work by the programmer.

27.3.5 Clone Detection

Based on the information provided by the parse tree we implemented an algorithm that detects clones in the source code [8]. It is an adaption of the algorithm for finding frequent itemsets.

27.3.6 Metrics

Different kinds of primitive source code metrics like the number of packages, the number of inner classes or the amount of comment in the source can be calculated. Additionally metrics like reuse ratio or sophisticated metrics like those proposed in [1, 5] can be calculated as well as experimental metrics.

27.4 Conclusion

We have described the architecture and the applications of *JTransform*, a general-purpose Java source code transformation framework. *JTransform* has been designed according to usual object-oriented design principles, well known design patterns have been used. It consists of a front end producing an annotated parse tree and various back ends.

Custom back end applications can simply be plugged in by implementing a tree visitor. An alternative architecture of the tool can transform the parse tree to an XML document [3]. Thence, user applications may use an appropriate XML format such as XSLT/XPath as a general purpose query and transformation language. The stability of *JTransform* itself was validated by a test suite based on [6], the source code of the Java library and different Java online courses.

Bibliography

[1] Chidamber S, Kemerer C. A Metrics Suite for Object Oriented Design *IEEE Transactions on Software Engineering* **20**(6):476–493.

[2] Code Conventions for the Java^{TM} Programming Language
`http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html`

[3] Eichelberger H, Wolff von Gudenberg J. Object-oriented Processing of Java Source Code Software Practice and Experience, 2004, to appear

[4] Eichelberger H, Wolff von Gudenberg J. On the Visualization of Java Programs, In *LNCS 2269: Software Visualization* S. Diehl (ed.) Springer: Berlin, 2002; 295–306

[5] Genero M, Piattini M, Calero C. Early measures for UML class Diagrams *L'Objet* 2000 **6**(4):29–35.

[6] Jacks (Jacks is an Automated Compiler Killing Suite)
`http://www-124.ibm.com/developerworks/oss/cvs/jikes/~checkout~/jacks/jacks.html`

[7] D. Lea Draft Java Coding Standard
`g.oswego.edu/dl/html/javaCodingStd.html`

[8] V. Wahler *Erkennung von Klonen in Java-Programmen mit Data-Mining-Techniken*, Institut f. Informatik, Universität Würzburg, Diplomarbeit, 2003