JMangler

A Framework for Load-Time Transformation of Java Programs

Michael Austermann
SCOOP GmbH, Im Ahlefeld 23
D-53819 Neunkirchen-Seelscheid
maustermann@scoop-gmbh.de

Pascal Costanza and Günter Kniesel
University of Bonn, Institute of Computer Science III
Römerstraße 164, D-53117 Bonn
{costanza | gk}@cs.uni-bonn.de

Contact: Günter Kniesel

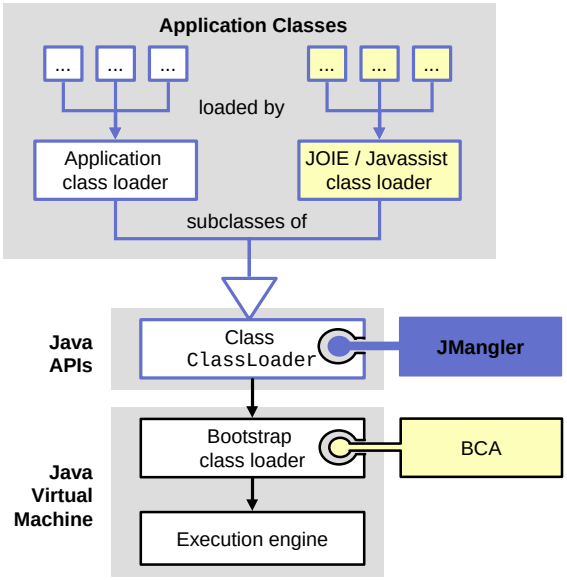
Motivation

Why class file transformation? Third-party libraries are usually only deployed in binary formats. Therefore, only class file transformations can capture the *entire* application. Fortunately Java's class file format is rich enough to allow for many powerful manipulations.

Why load-time transformation? Java allows classes to be loaded dynamically and further permits the name and contents of dynamically loaded classes to be determined at run-time, via reflection. There is therefore no way to determine, before run-time, the set of all classes that belong to a program. Thus the only point where it is possible to determine all classes that are actually used by a program and adapt them as needed is the dynamic class loading process.

Class Loader and JVM Independence

Current proposals for load-time transformation of Java classes, e.g. BCA [3], JOIE [2], or Javassist [1], are either dependent on the use of a specific class loader or dependent on a specific JVM implementation. JMangler follows a new approach that ensures class loader and JVM independence by providing a modified version of the class ClassLoader. Because the modified behaviour is enforced for every subclass of ClassLoader, JMangler is activated whenever an application-specific class is loaded. The following figure illustrates this approach.

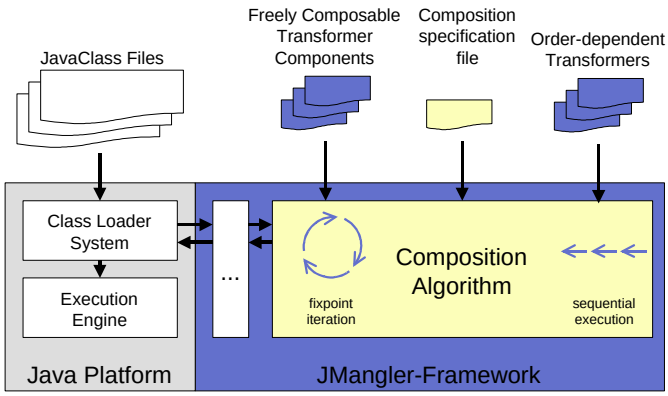


Composition of Transformers

In general, there is no satisfactory means to safely combine transformers that have been independently developed, without their being specifically designed for joint use. One problem of unanticipated composition is the possible occurrence of mutual dependencies among transformers or among transformed classes. In this case, applying each transformation only once may result in potentially incomplete programs. Therefore, transformations must be iterated until a fixed point is reached, that is, until no transformer requests any further changes. In this way, JMangler guarantees that every transformation is applied to each part of a program, even to parts added by other transformations.

Unfortunately, iteration of arbitrary transformations that have been composed in different orders yields different fixed points. However, we have shown that independent development and automatic composition is possible for the interface transformations that are supported by JMangler. JMangler's distinction between interface transformations and code transformations marks the border between freely composable transformer components and order dependent transformations.

JMangler's Architecture



This distinction between interface and code transformations is reflected in the transformation process which is partitioned into two phases. In the *first phase*, each interface transformer analyzes the target classes, decides which transformations are to be carried out, and requests these transformations from JMangler. The framework checks the validity of the requested transformations, chooses the order in which legal transformations are to be applied, and performs the transformations. This process is repeated until no further interface modification requests are issued. In the *second phase*, only code transformers are activated. They are executed exactly in the order indicated in the transformer configuration file that is passed as a parameter to JMangler.

Summary of Other Features

Powerful transformations JMangler allows for a wide range of code and interface transformations. *Code transformations* may be any changes to the bodies of existing methods. *Interface transformations* include addition of interfaces, classes, methods, fields, annotations, changes to the class hierarchy, and many others.

Support for multi-class transformers JMangler transformers can process *sets* of classes. This allows them to take complex dependencies between classes into account during the transformation process.

Support for multiple transformers Multiple transformer components can be applied simultaneously to all classes of a program.

Easy configuration Transformers can be combined just by editing an XML file.

Caching of transformed class files The output of the transformation process can be stored and reused for subsequent executions of the application. Transformations need to be reapplied only when the initial program is changed.

In-situ modifications A class is modified in situ rather than by the creation of wrapper classes to incorporate changes. This means that there are no performance penalties due to the transformation in the modified program.

Enforcement of binary compatibility Only changes that respect the Java Binary Compatibility Specification are allowed.

100% pure Java The JMangler framework is entirely written in Java and can be used on any platform supporting the JDK1.3. It has been tested on Linux, Windows, and Solaris.

Free availability JMangler is freely available under the terms of the GNU LGPL and can be downloaded from http:// javalab.cs.uni-bonn.de/research/jmangler/.

References

[1] Shigeru Chiba: *Load-Time Structural Reflection in Java*. In: E. Bertino (Ed.). ECOOP 2000 – Object-Oriented Programming. Proceedings, Springer, LNCS 1850, 2000.

[2] Geoff A. Cohen, Jeffrey S. Chase, and David L. Kaminsky: *Automatic program transformation with JOIE*. Proceedings of the USENIX 1998 Annual Technical Conference, Berkeley, USA, 1998. USENIX Association.

[3] Ralph Keller and Urs Hölzle: *Binary Component Adaptation*. In: E. Jul (Ed.). ECOOP '98 – Object-Oriented Programming. Proceedings, Springer LNCS 1445, 1998.

[4] Günter Kniesel, Pascal Costanza, and Michael Austermann: *JMangler – A Framework for Load-Time Transformation of Java Class Files*. Submitted to IEEE International Workshop on Source Code Analysis and Manipulation, Florence, Italy, November 10th, 2001.