



Newsletter



Editorial

Welcome to the new issue of our FUJABA newsletter. In this newsletter we report on the 1st International Fujaba Days and continue our series about realized and pending plug-ins for FUJABA. First of all, we start with an announcement. (rw)

IBM Eclipse Innovation Award

The Software Engineering Group at the University of Paderborn managed by Prof. Dr. Wilhelm Schäfer receives an Eclipse Innovation Award from IBM Corp. in the amount of \$15,000 USD. The contest was attended by 285 projects, of which 78 were selected for funding. The funding will be used to develop a learning environment for object-oriented concepts and object-oriented design based on the development environment Eclipse which is widely used within the IT industry.

The Software Engineering Group gained many years of experience with integrated development environments. Since 1997 the tool FUJABA has been developed at the group. It was successfully applied to teach object-oriented concepts in computer science courses at secondary schools in Paderborn and Braunschweig. The funding will be used to port these concepts to the integrated development environment Eclipse. (lw)

Fujaba Days 2003 Report

The 1st International Fujaba Days took place in Kassel, Germany on October 5-7, 2003. The main intent of the workshop was to bring together Fujaba developers and users to present their ideas and projects and to discuss them. Approximately 35 researchers, developers and users from Finland, Belgium, and England and from three different sites in Germany attended this event and contributed to a successful workshop.

The workshop was organized in three sessions, in which the speakers presented their work like e.g. real-time state charts, model checking, refactoring, document centered modeling and collaborative tool support. Additionally, the workshop was accompanied by practical tutorials concerning the development of plug-ins for Fujaba.

After a full day of technical presentations and inspiring tutorial sessions, attendees flocked to the conference dinner. It took place at a restaurant with local specialties, fantastic beer and exotic drinks. In this wonderful ambience the participants carried on renewing contacts and exchanging ideas far into the night.

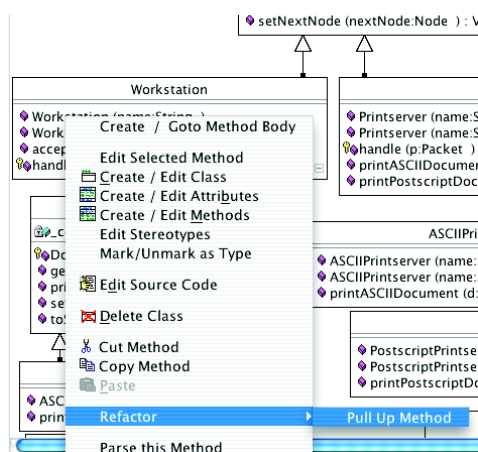
We thank the organizer Albert Zündorf and his team for a very interesting and well-organized event and are looking forward for the next International Fujaba Days. (rw)

Implementing Refactorings as Graph Rewrite Rules

Refactoring is a technique for improving the quality of existing software without affecting its external behavior. It is one of the cornerstones of Extreme Programming and is supported by leading IDEs such as IntelliJ IDEA and Eclipse. Without formal guarantees about the tool correctness, a refactoring user is required to write extensive tests.

Since reasoning about the correctness of a refactoring that is implemented in a language like Java is quite difficult, one may consider a program as a graph. The theory of graph rewriting can then be used as a formal basis.

A recent trend in software development is to generate code from visual specifications, as illustrated by OMG's MDA initiative. We have investigated how Fujaba can be used to generate Java refactoring implementations from visual graph transformations in the "Story Driven Modeling" (SDM) language. We can conveniently prove that our refactoring implementations preserve certain behavioral properties by reasoning at the graph level.

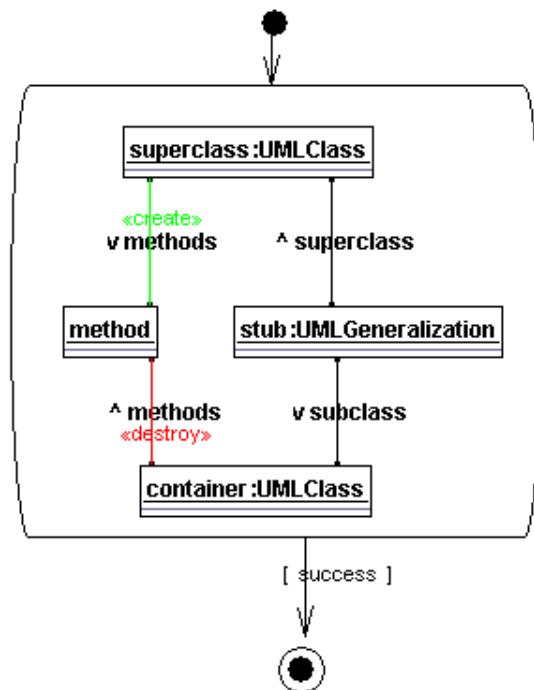


The initial prototype of the refactoring plugin implements the "Pull Up Method"

refactoring. By right-clicking on a method in a class diagram, a user can decide to move a method from its current class to the superclass. Behavior preservation is (at least partly) guaranteed by checking a refactoring's precondition. In the case of "Pull Up Method", for example, the signature of the method should not yet occur in the superclass.

Each refactoring inherits from the "Refactoring" interface, and implements the methods for checking the precondition and executing the transformation as SDM specifications.

PullUpMethod::execute (target: ASGElement): Void



Future versions of the plugin will make use of advanced SDM features such as transitive closure and negative application conditions. For more information, please contact Niels Van Eetvelde (niels.vaneetvelde@ua.ac.be). (nve, pvg)

Design Patterns Recognition with the Fujaba Tool Suite

When reverse engineering software systems the first important task is to understand the systems source code. Design patterns are used in forward engineering as good solutions to recurring problems and form a common vocabulary among developers. Thus, recognizing implementations of design patterns in existing software systems helps the reverse engineer to understand the system.

The description of design patterns is usually informal to a large extent. For semiautomatic recognition of design patterns they have to be formally specified. We developed a pattern specification language based on graph grammars and an iterative and scalable inference algorithm.

Pattern Specification

The PatternSpecification plug-in provides a graphical editor for the specification of patterns. We specify patterns as graph transformation rules, with respect to the abstract syntax graph (ASG) of a systems source code. Applying pattern rules results in enriching the ASG with annotation nodes which may be linked to an arbitrary number of ASG elements. Thus, annotation nodes mark pattern implementations recognized by pattern rules.

A pattern rule is defined by a left-hand side (LHS) and a right hand side (RHS). The LHS of the rule describes the structure that has to be found in the ASG if an instance of the pattern exists. The LHS may also contain annotations created by other pattern rules thereby permitting a composition of rules. Rules requiring annotations created by other rules depend on those rules. Furthermore, in the LHS special ASG elements or annotations are designated which trigger the

execution of the rule. Triggers and dependencies between rules are used by the inference engine to determine the rule execution order (see below).

The right-hand side (RHS) of a pattern rule defines an annotation node and links to certain ASG elements to be created when the LHS could be matched, i.e., an implementation of the pattern could be found.

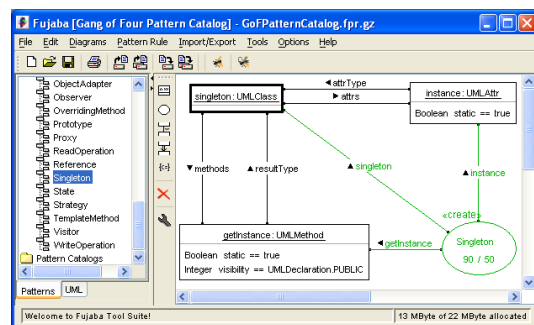


Fig. 1: A pattern rule in the Fujaba Tool Suite

Each pattern rule is specified in a separate diagram (cf. Fig. 1). The plug-in provides additional pattern catalogue diagrams which display dependencies and trigger relationships between rules. A catalogue of rules is translated into pattern recognition engines used by the pattern inference.

Pattern Inference

The pattern rules are applied by an inference algorithm which is implemented by the InferenceEngine plug-in. After parsing the source code of the system into an ASG the inference engine is started by loading a catalogue of pattern rules.

The inference engine uses a pattern dependencies net (PDN) in which pattern rules are organized in levels according to their dependencies and trigger relationships. Based on the PDN the inference engine applies rules scheduled in priority queues. It starts with rules that are independent from other rules.

Successfully applied rules create annotations which in turn trigger other rules at higher levels. This is called the bottom-up mode of the inference engine. Newly triggered rules are scheduled according to their levels in descending order, i.e., higher level rules which produce meaningful results are executed as early as possible.

Scheduled higher level rules may depend on other rules that have not been applied yet, i.e., rules may require annotations created by lower level rules. In this case the inference engine switches from bottom-up to top-down mode and tries to recursively establish all missing annotations by applying the appropriate rules.

The inference engine works semi-automatically because it involves the reverse engineer in the analysis. The reverse engineer may pause the inference at any time and inspect the results produced so far. Furthermore the engineer may modify or manually add (hypothetical) results and continue the inference. The changes are then considered in the further analysis.

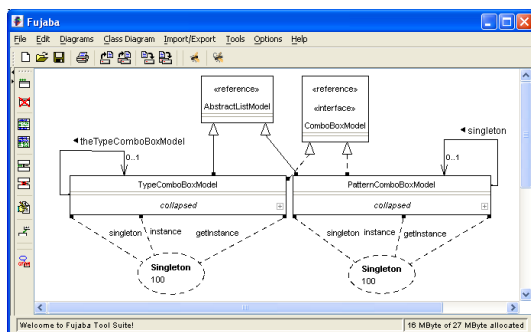


Fig. 2: Analysis results displayed in a class diagram

The analysis results, i.e., the annotations, are displayed in class diagrams which can be directly obtained from the ASG. Fig. 2 shows a screenshot of the Fujaba Tool Suite with an annotated class diagram after an analysis with the inference engine.

Current and Future Work

Our current work on the one hand focuses on the recognition of Anti Patterns as a basis for quality evaluation and refactoring. On the other hand we extend our analysis to dynamic behaviour to distinguish between statically similar but dynamically distinguishable design patterns.

For further details have a look at our web site, especially on the reverse engineering project. There you can find publications and in the near future a downloadable version of the Fujaba Tool Suite RE (Reverse Engineering) edition. (lw, mm)

May I introduce...



... Jörg Niere, one of the first three master's thesis candidates who worked on Fujaba. Together with Lars Torunski and Thorsten Fischer he developed the first Fujaba version under

the supervision of Albert Zündorf in 1998.

Between 1998 and 2003 Jörg Niere worked as a research assistant in the software engineering group of Prof. Wilhelm Schäfer at the University of Paderborn. In this time his main research activities have been reverse engineering and especially design patterns recognition based on source code.

Today, Jörg Niere works as an assistant professor in the group of Prof. Udo Kelter at the University of Siegen, Germany. He is still working with Fujaba on automation engineering especially on the specification of information systems closely coupled with micro controllers. (lw)