Model-driven Monitoring: Generating Assertions from Visual Contracts

Marc Lohmann, Gregor Engels
University of Paderborn
Department of Computer Science
Warburger Str. 100, 33098 Paderborn, Germany
engels, mlohmann@uni-paderborn.de

Stefan Sauer
Software Quality Lab (s-lab)
University of Paderborn
Warburger Str. 100, 33098 Paderborn, Germany
sauer@s-lab.upb.de

Abstract

The Visual Contract Workbench is a tool that supports model-driven development of software systems by lifting the Design by Contract idea, which is usually used at the code level, to the model level. It uses visual contracts for graphically specifying the pre- and post-conditions of an operation. Java classes with JML (Java Modeling Language) assertions are generated from visual contracts to facilitate automatic monitoring of the correctness of the programmers' implementation.

1 Introduction

Design by Contract (DbC) [4] is a powerful technique for creating reliable software. The basic idea of DbC is that the relationship between a class and its clients is specified by a contract, a kind of formal agreement expressing each party's rights and obligations. The client must guarantee certain conditions before calling an operation (pre-condition), and in return the class guarantees certain properties that will hold after the operation's execution (post-condition).

Regularly, contracts are specified in an extension of the programming language for logic formulae and embedded in the program code. They are translated by a compiler into executable code. This code checks the fulfillment of the pre- and post-conditions when a client calls an operation. Any violation of a contract can so be detected while the program is executed. Thus, it is possible to monitor whether a program behaves correct according to its specification.

However, describing a specification by using the programming language itself is not adequate in today's model-driven software development processes. We propose to use pairs of UML composite structure diagrams for specifying the pre- and post-condition of an operation. Each pair constitutes a visual contract. Additionally, we have defined a transformation of our visual contracts into JML (Java Modeling Language) assertions [3]. JML [2] extends Java with DbC concepts. The generated JML assertions can be used to monitor whether the manually coded program behaves according to its specification by visual contracts. Thus, we enable *model-driven monitoring*.

In this paper, we present how a new tool, called *Visual Contract Workbench*, can be used to embed our approach in a model-driven software development process.

2 Model-driven Monitoring Approach

Our approach lends itself to model-driven software development processes. Visual contracts are interpreted as models of behavior from which code for runtime assertion checking can be generated. The visual contracts also specify the behavior which has to be implemented by programmers. We exemplify how to enable model-driven monitoring in a software development process by using the Visual Contract Workbench.

In the first step, a software designer uses the Visual Contract Workbench to build a design model of the system. This model consists of a class diagram which is complemented by visual contracts. The class diagram describes the static aspects of the system. Each visual contract specifies the behavior of an operation. The behavior of the operation is given in terms of data state changes by pre- and post-conditions, which are modeled by pairs of composite structure diagrams (Fig. 2). Both the pre- and post-condition of a visual contract are typed over the design class diagram.

In the next step, the workbench can be used to generate Java code from the design model. This generation process consists of two parts. First, we generate Java class skeletons from the design class diagram. Second, we generate JML assertions from every visual contract and annotate the corresponding operations with their JML contracts. The JML assertions allow us to validate the consistency of models

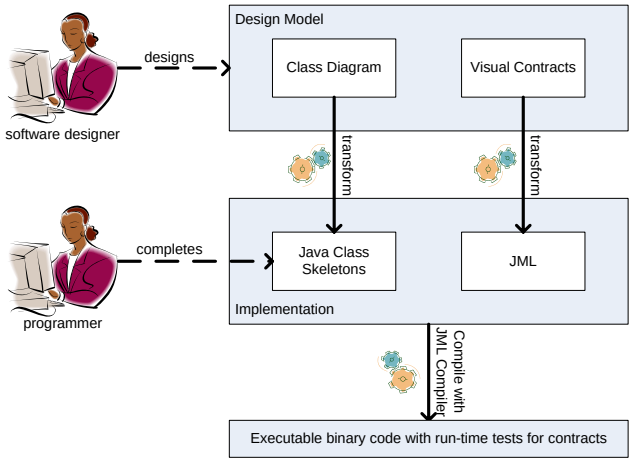

Figure 1. Overview of the Visual DbC methodology

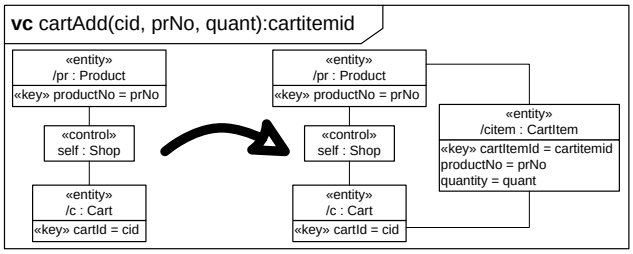

Figure 2. Example of a visual contract

with manually derived code. The execution of such checks is transparent in that, unless an assertion is violated, the behavior of the original program remains unchanged.

Then, a programmer uses the generated Java fragments to fill in the missing behavioral code in order to build a complete, functional application. Her programming tasks emanate from the design model of the system. Particularly, she will use the visual contracts as reference for implementing the behavior of operations. She has to code the method bodies, and may add new operations to existing classes or even completely new classes, but she is not allowed to change the JML contracts. The latter guarantees that the JML contracts remain consistent with the visual contracts.

Our approach builds upon a loose semantic interpretation of visual contracts [1]. They are interpreted as a minimal description of the data state transformation which has to be implemented by the programmer. Thus, a visual contract specifies only what at least has to happen on a system's state, but it allows the programmer to implement additional effects. This loose interpretation is necessary both to give the programmer the opportunity for optimizing her code, e.g. by adding new classes or methods, and to generate assertions from partial, incomplete models.

When a programmer has implemented the behavioral code, she can start the JML compiler from the workbench to build executable binary code. This binary code consists of the programmer's behavioral code and additional executable runtime checks which are generated by the JML compiler from the JML assertions. The runtime checks monitor the pre- and post-conditions during the execution of the system. They monitor whether the manually coded behavior of an operation fulfills its JML specification. Thus, we indirectly check whether the behavioral code complies with the visual contract specification of the design model since the JML assertions are purely generated from the visual contracts. That means, we support a model-driven monitoring of implementations by transforming our visual contracts into JML contracts.

3 Conclusions

We have developed a model-driven monitoring methodology. Visual contracts are introduced as a technique for specifying the pre- and post-conditions of an operation by pairs of UML composite structure diagrams. By using the UML, we build on a well-known standard that is predominantly used in today's model-based development methods. Our methodology also supports generation of code by a translation of visual contracts into the Java Modeling Language, a Design by Contract extension for Java.

For an efficient deployment of our methodology, we provide a Visual Contract Workbench. This Eclipse plug-in allows developers to coherently model class diagrams and visual contracts. The workbench is complemented by code generation facilities for Java classes with assertions for their operations.

Our methodology is currently considered by a industrial partner software company of the s-lab for deployment in their software development projects.

References

[1] G. Engels, M. Lohmann, S. Sauer, and R. Heckel. Model-driven monitoring: An application of graph transformation for design by contract. In *International Conference on Graph Transformation ICGT 2006*, September 2006.

[2] G. T. Leavens, A. L. Baker, and C. Ruby. Preliminary design of JML: A behavioral interface specification language for Java. Technical Report 98-06-rev27, Department of Computer Science, Iowa State University, February 2005.

[3] M. Lohmann, S. Sauer, and G. Engels. Executable visual contracts. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 63–70, 2005.

[4] B. Meyer. Applying "Design by Contract". *IEEE Computer*, 25(10):40–51, 1992.