Cate: A System for Analysis and Test of Java Card Applications

Peter Pfahler and Jürgen Günther
Email:peter@uni-paderborn.de jguenther@orga.com
Universität Paderborn, Department of Computer Science, D-33098 Paderborn
ORGA Kartensysteme GmbH, Am Hoppenhof 33, D-33104 Paderborn

Abstract: Cate is a domain-specific testing environment. It integrates both static and dynamic analyzes that are designed for Java Card application software. Cate supports the test process by analyzing the command/response behavior of the software, by performing test coverage analysis and by providing tools to visualize the analysis results. This paper gives a concise overview over the system which is successfully employed in the area of smart card development for mobile phones.

1 Introduction

The importance of quality assurance is particularly high for the development of smart card applications. On one hand, smart cards are frequently applied in areas with increased security requirements, like banking or subscriber identification for mobile phones. On the other hand, detecting defects in smart card software will cause immense costs for exchanging large quantities of issued cards.

Java Card [JCR99] [SUN97] [JCV99] is a technology where the smart card contains a Java virtual machine. Card applications are developed in a subset of the Java programming language. They are compiled to Java byte-code, installed on the card and executed by that virtual machine at runtime.

Using Java Card technology means that program analysis tools [Thi02] [Thi01] that already exist for the standard Java language can be used in supporting quality assurance for Java card applications. Our goal to have a system that supports both the static and dynamic analysis of Java Card applications led to the development of the Cate system. Cate employs specialized static analyzes that take advantage of typical coding structures in Java Card applications. They determine the control flow paths that are taken for different commands sent to the smart card. In its dynamic subsystem Cate supports card software testing by instrumentation-based coverage analysis. Finally, as a special feature, Cate combines the result of static and dynamic analyzes by computing the command words that have to be issued to the smart card to reach the areas of code which have not yet been covered by test cases. The name “Cate” stands for “Card Analysis and Testing Environment”.

The rest of this paper is structured as follows:

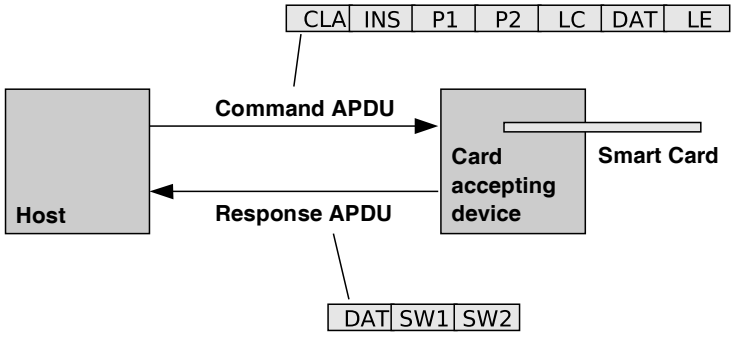

Figure 1: The smart card communication model

Section 2 discusses static and dynamic program analysis for Java Card applications. We will give a short introduction to the general idea behind static and dynamic program analysis before we present our approach of pattern based static analysis of Java Card applications. Finally, the dynamic test coverage analysis is discussed together with the test support that combines the results of static and dynamic analyzes.

Section 3 presents the Cate System. Cate was developed cooperatively in a project by the University of Paderborn and the ORGA Kartensysteme GmbH. We will introduce its central features and discuss the typical work-flow using the system. Section 4 reports on our practical experience using the Cate system. We present common usage scenarios and describe the typical error situations which have been detected using Cate.

2 Static and dynamic program analysis for Java Card Applications

Static program analysis examines a program by inspecting the program code while dynamic analysis gains information by observing a program being executed.

Our static analysis for Java Card applications works on Java byte-code. The advantage of examining byte-code instead of source code is that applications can be analyzed even if their source code is not available. This is often the case for library code.

The goal of Cate's static program analysis is to gain information about the so-called command/response behavior of Java Card applications. The term "command/response" reflects the typical working of smart card applications: A command is sent to the card through the card accepting device (CAD). The card reacts by executing certain methods depending on the command and its parameters. The card application finishes by returning a certain response code, possibly accompanied by some computed data [Che00]. Fig. 1 illustrates this smart card communication model.

While in general it is hardly possible to statically predict the runtime I/O behavior of a program, our command/response analysis makes use of the typical structure of card applications: execution starts in an entry method called `process`. This method gets the issued command as a so-called APDU ("application data unit") as a parameter. Usually it

switches control flow depending on certain fields of the APDU calling different methods for command processing. The program analysis we implemented in the Cate system is based upon these typical building blocks (cliches) of card applications: Accessing APDUs, e.g. to assign them or pass them as parameters, accessing the fields of an APDU, like instruction byte or parameters, or comparing APDU fields to constants to determine the control flow. Cate's control and data flow analysis [Pan00] [Muc97] determines which APDU leads to the execution of which control flow path. By additionally determining which resulting status code is returned from the basic blocks, the Cate system is able to compute a complete list of command/result combinations. This result can directly be compared to the original specification to determine mismatches between specification and implementation.

The central goal of Cate's dynamic analysis subsystem is to support testing by automatic test coverage analysis. Test coverage analysis is a structural testing technique that finds out how much of an application is "covered" by a given set of test cases [Bal98] [MLKS94]. In our case it is implemented by instrumenting each basic block of an Java Card application to record its execution during a test run [BL96]. Similar to static analysis our instrumentation works on the byte-code level, such that source code does not have to be available to measure coverage.

Cate does not require the applications to be executed on real Java Cards. It also provides interfaces to the Java Card Simulator JCWDE ("Java Card Workstation Development Environment") that comes with the Sun Java Card Development Kit and other proprietary simulation tools.

Cate's test coverage analysis results in two coverage measures: basic block coverage ("C0") and path coverage ("C1"). Both metrics are easily computed from the profile data produced by executing the instrumented code. Nevertheless, they are very helpful for test engineers.

In addition to computing the coverage measures, the Cate system can visualize the control flow through the Java Card application. By coloring edges (control flow branches) and nodes (basic blocks) according to their coverage status, it provides valuable testing support. Fig. 2 shows an example graph.

As a unique feature, the Cate system is finally able to combine the results of the pattern based static command/result analysis with the code coverage findings during dynamic analysis: The control flow graph visualization that is computed by the static analysis subsystem is enhanced by adding predicates to control flow edges. These predicates are expressed in terms of APDU components and specify the conditions under which a certain control flow edge is taken. Thus the testing engineer gets valuable support in enhancing his test cases by using additional command APDUS to increase code coverage. In section 4 we will shortly discuss how this combination of static and dynamic analyzes can even positively influence the cooperation between software developers and testing engineers.

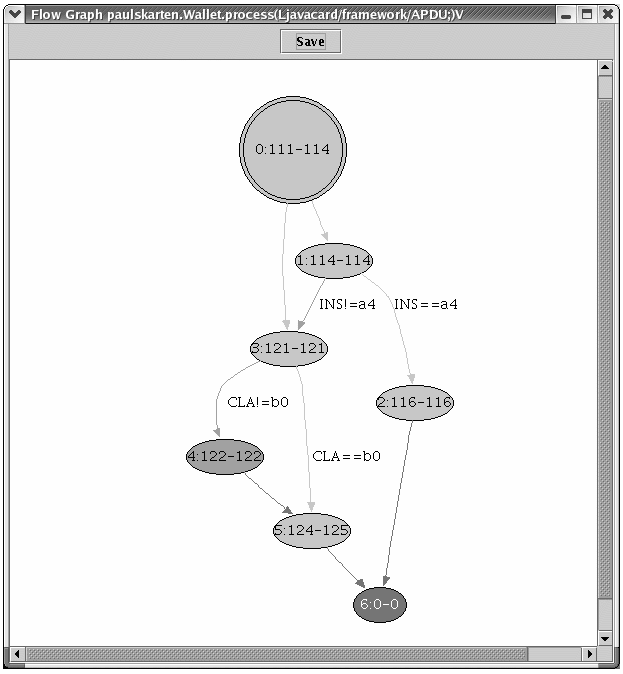

Figure 2: A colored control flow graph: node 4 is colored red since it has not been executed during testing.

3 The Cate System: integrating static and dynamic analysis tools

Cate is designed as a working environment for Java Card software tests. Fig. 3 gives a first impression or Cate's user interface. The upper half of Cate's screen is dedicated to the static analysis phase. In the upper left window there is a source structure browser that represents the typical tree view for object oriented Java programs. The right window displays the source code for the selected Java Card method. The static analysis menu provides an entry to display the complete list of command/response combinations.

The lower half of the screen contains the dynamic analysis tools. Java Card simulators can be initialized or shut down. Test suites, test cases, or single commands can be selected for execution. The lower right window displays protocols of the APDU execution. Computed results are compared to their expected values (if the test cases provide them). During test case execution, the source code display in the upper right window gets colored to reflect the coverage of the source code.

Context menus provide additional information: for each Java Card method the test coverage measure can be displayed. In addition to the source code which is colored according to its coverage status, a colored control flow graph is provided for a more high-level view on test coverage. On the dynamic side, Cate provides information for each executed APDU

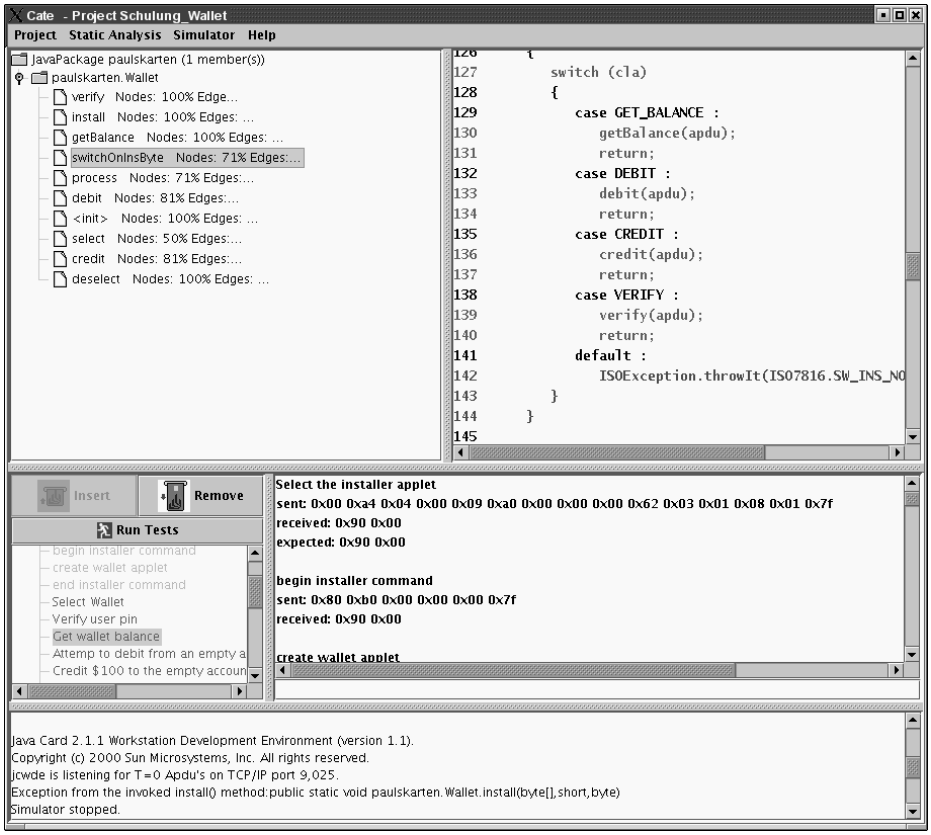

Figure 3: The Cate System, showing source code browser, colored source code, test case selector, test script execution log and system log window.

which control flow path has been taken due to this APDU. The protocol window can be turned into an editor to directly change or enhance test cases to increase coverage.

4 Practical Experience Summary and Conclusion

The central task of Cate's static analysis tools is to produce a list of command/response combinations. This information can easily be used to compare the implementation's behavior with the specification. While this is a very valuable tool to have, its practical relevance turned out to be limited due to the fact that traditionally smart card applications are developed very carefully, making e.g. extensive use of code reviews. Thus, the mismatches between specification and implementation were usually eliminated before the Cate system was applied.

Cate's dynamic analyzes showed a greater impact on the development and testing process.

Typically software and test cases are developed based on a functional specification. Two well-known problems are inherent to this procedure:

- The test cases may not cover a required function such that parts of the software are excluded from testing.
- The software may contain parts that are not justified by the specification.

Cate's dynamic coverage analysis equally detects both kinds of problems. In practice, both the test engineer and the developer then have to look at the code to discover the reason for the uncovered areas. This leads to an "implicit" code review of critical software parts. In contrast to a "standard" code review process the implicit one lacks the perception of a personal supervision as it primarily aims at the cooperative solution of a concrete problem. The psychological effect of this difference on the work of testers and developers is not to be underestimated.

In our practical experience code not covered by test cases has turned out to be mainly caused by double security checks. Although this normally does not influence the functionality directly, such code consumes execution time and memory, i.e. resources which are always quite limited on smart cards.

Cate produces a statement about the quality of the test suite with measurements being based on the implementation of the software. Furthermore, by combining test coverage results and control flow analysis, the system is able to give valuable hints for improving the test suite. Thus the process not only results in the delivery of high quality software but also in a reduction of the overall development time and costs.

Finally, using the Cate system to make testing more safe has shown an additional positive effect on the structure and readability of the code itself. This effect is caused by both the overview that the developers gained from our visualization tools and the implicit code reviews to find out the reasons for uncovered code. In some cases the findings led to an improved software structure, which clearly increased the maintainability of the investigated smart card code.

References

[Bal98] Helmut Balzert. *Lehrbuch der Softwaretechnik, Teil 2: Softwaremanagement, Software-Qualitaetssicherung, Unternehmensmodellierung*. Spektrum Akademischer Verlag, Heidelberg, Berlin, 1998.

[BL96] Thomas Ball and James R. Larus. Efficient Path Profiling. In *International Symposium on Microarchitecture*, pages 46–57, 1996.

[Che00] Zhiqun Chen. *Java Card technology for Smart Cards: architecture and programmer's guide*. Java series. Addison-Wesley, Reading, MA, USA, 2000.

[JCR99] SUN Microsystems, Inc. *Java Card 2.1 Runtime Environment (JCRE) Specification*, February 1999. Final Revision 1.0.

[JCV99] SUN Microsystems, Inc. *Java Card 2.1 Virtual Machine Specification*, March 1999. Final Revision 1.0.

[MLKS94] Y. Malaiya, N. Li, R. Karcich, and B. Skbbe. The relationship between test coverage and reliability, 1994.

[Muc97] Steven S. Muchnick. *Advanced compiler design and implementation*. Morgan Kaufmann Publishers, 2929 Campus Drive, Suite 260, San Mateo, CA 94403, USA, 1997.

[Pan00] Robert Panzer. Programmanalyse zur Qualitätssicherung von Kartenanwendungen in Java. Master's thesis, Universität Paderborn, Germany, December 2000.

[SUN97] SUN Microsystems, Inc. *Java Card 2.0 Language Subset and Virtual Machine Specification*, revision 1.0 final edition, October 1997.

[Thi01] Michael Thies. *Combining Static Analysis of Java Libraries with Dynamic Optimization*. Dissertation. Shaker Verlag, ISBN: 3-8322-0177-7, April 2001.

[Thi02] Michael Thies. Annotating Java Libraries in Support of Whole-Program Optimization. In *Proceedings of Workshop on Intermediate Representation Engineering for the Java Virtual Machine (IRE-2002)*, Dublin, Ireland, June 2002.