Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 113 (2005) 217–233
www.elsevier.com/locate/entcs

Guaranteeing Correctness Properties of a Java Card Applet

Lars-Åke Fredlund

Box 1263, 164 29 Kista,
Swedish Institute of Computer Science, Sweden
e-mail: fred@sics.se

Abstract

The paper describes an experiment in which a framework for model checking Java byte code, combined with the application of runtime monitoring techniques through code rewriting, was used to guarantee correctness properties of a Java Card applet.

Keywords: Java byte code, runtime monitoring, code rewriting, Java Card.

1 Introduction

The Java Card platform [8] is a platform for building multi-application smart cards. It is based on a subset of Java which omits features such as concurrency through threads, garbage collection, and many API functions. However, to support multiple applications co-existing on the same card (e.g., both a purse applet and a loyalty applet), there is a notion of an applet. Java Card applets are implemented by extending the Java Card API class `javacard.framework.Applet`. Briefly an implementation is required to provide a method `install` which is called upon installation of an applet, methods `select` and `deselect` for selection/deselection of a particular applet on a card, and the "main" method `process` which is called by the card runtime environment (operating system) upon receiving an event from the card environment intended for that applet. An applet can also implement a method `getShareableInterfaceObject` for permitting other applets on the same card to call it.

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2004.01.033

Unfortunately the Java Card programming platform provides weak support for separating applets. For instance nothing in the standards prevents a malicious or badly written applet from allocating all persistent memory on a card (the little there is), and since the standard does not require garbage collection this is a very undesirable state-of-affairs. Similar concerns exists for inter-applet calls, although they are controlled by a rather weak firewall mechanism. Thus there are significant dangers with permitting new applets onto a functioning smart card, and as a result one of the chief innovations of Java Card, i.e., multiple applications co-existing on the same card, is in practise not used much at all.

To improve upon this situation the formal design techniques group of SICS have been using fully automatic and low-cost (in terms of execution speed and memory usage) verification methods that could, potentially, be used by an on (or off) card runtime system to determine at load time whether a new applet should be permitted onto a card with pre-existing applets or not. In a first experiment, reported in [3], we analysed inter-method calls of multi-applet Java Card smart cards using model checking of Java byte code. In this paper we extend the treatment to memory allocation concerns. In case the safety of an applet cannot be proved using model checking, we as a complementary technique instrument a compiled applet with a runtime monitor to guarantee that it adheres to a safe memory allocation policy.

To provide a semantics foundation for the analysis of Java Card applets we use the abstract notion of a program graph, capturing the control flow of programs with procedures/methods, and which can be efficiently computed. The behaviour of such program graphs is defined through the notion of pushdown systems, which provide a natural execution model for programs with methods (and possibly recursion), and for which completely automatic model checkers for LTL exist, e.g., Moped [7]. The details of the translation are elaborated in section 3.1, and sections 3.2, 3.3 and 3.4 describes the logic and our use of the Moped model checking tool in further detail.

The example considered in the paper is a real applet submitted by Schlumberger. The applet is monolithic, and does not communicate with other applets. In section 4 we formalise and attempt to verify the property that no memory is allocated by an applet (after a personalization process) using model checking. As the satisfaction of this property is shown to depend critically on properties of data, which requires reasoning using less coarse abstractions than the ones implemented in the call graph extraction tool, we consider in section 5 the complementary use of runtime monitoring techniques to guarantee the property.