A Java Card CAP converter in PVS [1]

Thomas Genet [2] Thomas Jensen [2] Vikash Kodati
David Pichardie [2]

IRISA
Université de Rennes 1 & CNRS
Campus de Beaulieu
F-35042 Rennes

Abstract

The Java Card language is a trimmed down dialect of Java aimed at programming smart cards. Java Card specifies its own class file format (the Java Card Converted APplet (CAP) format) that is optimised with respect to the limited space resources of smart cards. This paper deals with the certified development of algorithms necessary for the conversion of ordinary Java class files into the CAP format. More precisely, these algorithms are concerned with constructing and compressing method tables and constant pools. The main contribution of this paper is to specify and prove the correctness of these algorithms using the theorem prover PVS.

1 Introduction

The Java Card language [7] is a trimmed down dialect of Java aimed at programming smart cards. As with Java, Java Card is compiled into bytecode, which is then verified and executed on a virtual machine [4], installed on a chip on the card itself. However, the memory and processor limitations of smart cards necessitate a further stage, in which the bytecode is optimised from the standard class file format of Java, to the *CAP file* format [8]. The core of this optimisation is a *tokenization* in which names (strings) are replaced with tokens (integer values). Replacing strings with integers reduces the size of the code and enables a faster lookup of virtual methods using the standard object-oriented technique of *vtables*. Additional optimisations are obtained by a *componentisation* that merges class files from the same package into one *CAP file*. This means that data which is common to several class files can be

[1] This work was partially funded by the European IST R&D project 2000-26328 "Verificard"

[2] Email: {Thomas.Genet, Thomas.Jensen, David.Pichardie}@irisa.fr

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

shared and that symbolic references between classes from the same CAP file can be transformed into memory offsets relative to the CAP file.

In a companion article [3] we have described a semantic framework for proving the correctness of Java Card tokenization. The basic idea underlying that framework was to give an abstract description of the constraints from the official specification of the tokenization and show that any program transformation satisfying these constraints is 'correct'. Notice that this is independent of showing that there actually exists a collection of functions satisfying these constraints (which is not done in *op. cit.*). The main advantage of decoupling 'correctness' into two steps is that we get a more general result: rather than proving the correctness of one particular algorithm, we are able to show that the constraints described in Sun's official specification [8] (given certain assumptions) are sufficient.

The aim of the work reported here is to construct a provably correct program that will transform Java Card class files into equivalent CAP files—we will call such a program a *CAP converter*. The result mentioned above reduces considerably the proof obligations for constructing a provably correct converter. For the tokenization, it is enough to verify that the CAP converter respects constraints on the tokenization imposed in the official language definition (see 4) to ensure correctness. For the componentisation, the proof is facilitated by first developing an abstract theory of merging tables and then instantiating this theory to the relevant Java Card structures such as the constant pool (see 5). We develop the algorithm and proofs on a simplified model of Java Card programs, but the methodology remains valid for a full Java Card model.

The paper is structured as follows. We first provide an intuitive explanation of the purpose of tokenizing Java Card class files by describing the differences between method resolution for Java Card class files and for Java Card CAP files (Section 2). We then proceed (Section 3) to describe the PVS formalization of the class file and the CAP format on which the CAP converter operates. Section 4 presents the development and accompanying proofs of the tokenization part of the converter. Section 5 on componentisation describes the specification and the implementation of constant pool merging in the CAP format.

2 Virtual method dispatch in Java Card

In the Java language, when calling a method m on an object of class c, the bytecode is found using a method lookup function:

$$\texttt{lookup} : \texttt{Class\_ref} \times \texttt{Method\_ref} \rightarrow \texttt{Class\_ref} \times \texttt{Bytecode}$$

which implements the resolution of virtual method invocation. It takes a class reference c (the actual class of the receiver object), a method reference m (the signature and the class in which the m is declared), and returns the reference