

Enhancing Class Files: A Migration Path to Better Mobile-Code Representations

Michael Franz
University of California, Irvine
franz@uci.edu

Abstract

Despite its apparent flaws, the Java Virtual Machine's bytecode format (“Java bytecode”) is the established standard for distributing mobile programs across the Internet. Any proposal for an alternative better mobile-code format appears doomed from the start in light of the vast amount of already existing Java bytecode.

Somewhat surprisingly then, we are building a system in which Java bytecode is supplanted by an “enhanced” alternative representation, in a manner that is completely invisible to the end-user. The substitution occurs at the web server and is provided as a value-added service. When communicating with an enhanced client computer, the enhancement-aware server will send it an enhanced class file rather than a standard Java bytecode file. Enhanced clients, on the other hand, can process both kinds of class file, but will perform better using the enhanced version. Enhanced class files are generated from standard Java bytecode as an off-line process at the server.

1. Introduction

Java bytecode has become the de facto standard for transporting mobile code across the Internet. However, it is generally acknowledged that Java bytecode is far from being an ideal mobile code representation—a considerable amount of preprocessing is required to convert Java bytecode into a representation more amenable to an optimizing compiler, and in a dynamic compilation context this preprocessing takes place while the user is waiting.

Further, due to the need to verify the code’s safety upon arrival at the target machine, and also due to the specific semantics of the Java Virtual Machine’s particular security scheme, many possible optimizations cannot be performed safely in the source-to-Java bytecode compiler, but can only be done at the eventual target machine.

For example, information about the redundancy of a type check may often be present in the front-end (because the compiler can prove that the value in question is of the correct type on every path leading to the check), but this fact cannot be communicated safely in the Java bytecode stream and hence needs to be re-discovered in the just-in-time compiler. By “communicated safely”, we mean in such a way that a malicious third party cannot construct a

mobile program that falsely claims that such a check is redundant.

As a consequence, when Java bytecode is transmitted to another site, each recipient must repeat most of the analyses and optimizations that could have been performed just once at the origin if a better alternative mobile-code representation had been used instead.

In previous work, we have identified several such alternative mobile code representations [1, 2, 3]. These alternatives overcome many of the limitations of the Java bytecode language. Moreover, while providing the identical security guarantees as the Java Virtual Machine, they express most of them statically as a well-formedness property of the respective encoding itself. This eliminates the need for an expensive dataflow analysis at the code recipient’s site, greatly reducing the effort required for code verification.

That leaves the question how one would go about replacing the existing Java bytecode on the Internet with one of these superior formats—considering how much Java bytecode is actually out there, one would think this to be an unrealistic proposition. Surprisingly, it is precisely this problem that we present a solution to.

2. Architecture

In our architecture (Figure 1), class files containing ordinary Java bytecode and “enhanced” class files containing a more advanced intermediate representation coexist side-by-side. “Enhancement” is offered as a value-added service by certain web servers in this system,

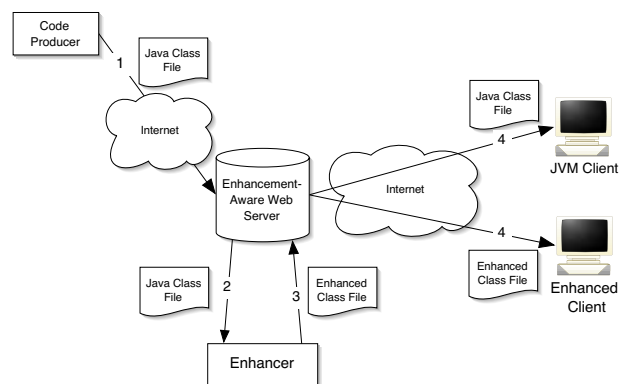


Figure 1. Flow of class files through the system

the beneficiaries are certain “enhanced” clients. One could imagine a situation in which a vendor equips all the workstations on a customer’s intranet with enhanced client software while installing an “enhancer” at the central server. This would boost performance of day-to-day operations within the corporate intranet, while coincidentally also benefiting any *external* enhanced clients of the server that might exist additionally, as well as any transactions of *internal* clients with *external* enhanced servers.

Several flows of class files are identified in Figure 1: all class files originate in the standard Java bytecode format and are placed on a server for hosting (1). Some of the hosting servers will provide an *enhancer* that will input an ordinary Java bytecode file (2) and generate an enhanced class file from it (3). Client computers negotiate with every server they connect to; if an enhancement-aware server detects an enhanced client, it will send it an enhanced class file if one is available; otherwise, it will send the standard Java bytecode file (4).

An enhanced client, on the other hand, can process both regular Java bytecode files as well as enhanced class files (Figure 2). This enables it to communicate with all servers on the Internet. If it is communicating with an enhancement-unaware web server, or if no enhanced class file is available on an enhancement-aware server, then it will fall back onto the classic Java bytecode format. If an enhanced class file is available, then it will be used instead, resulting in a higher level of performance.

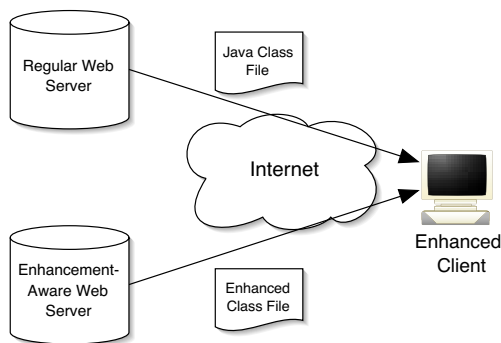


Figure 2. Enhanced clients can run both ordinary JVM class files as well as enhanced class files

In our prototype implementation [4] of an enhanced client, we augmented an existing dynamically optimizing Java virtual machine with the necessary extensions enabling it to also process enhanced class files in our own *SafeTSA* mobile-code format [2]. In our implementation, the two formats actually share the identical low-level code generator that translates from a low-level intermediate representation (LIR) to the final native instruction stream, resulting in comparable final code quality for the two formats when compilation time is unbounded.

The key point, however, is that compilation time in dynamic-compilation environments is hardly ever unbounded. In this situation, the enhanced class file

format has a substantial advantage, because fewer steps and less complex operations are needed (Figure 3) to verify and preprocess it into the LIR (a variant of Static Single Assignment Form in this case). The time that has thus been saved (essentially by performing analyses at the code producer’s site and transmitting the results within the enhanced mobile-code format in a tamper-proof manner) can then be expended on high-quality code optimization.

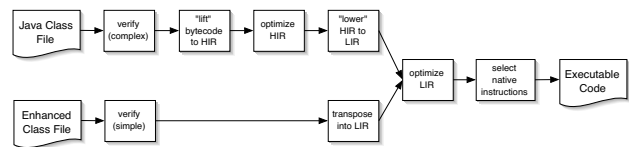


Figure 3. Enhanced class files require significantly reduced effort for verification and code generation; when compilation time is constrained, this means that better code can be generated in equal available time

Acknowledgement

The mobile-code research described here is a joint effort of a large research group, incorporating contributions from Wolfram Amme, Matthew Beers, Niall Dalton, Michael Franz, Peter H. Fröhlich, Vivek Haldar, Peter S. Housel, Chandra Krintz, Jeffery v. Ronne, Christian H. Stork, Ning Wang, and Sergiy Zhenochin.

Parts of this effort are sponsored by the National Science Foundation under grant CCR-9901689, and by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-1-0536.

References

- [1] W. Amme, N. Dalton, P. Fröhlich, V. Haldar, P. S. Housel, J. v. Ronne, Ch. H. Stork, S. Zhenochin, and M. Franz. “Project transPROse: Reconciling Mobile-Code Security With Execution Efficiency”, in *The Second DARPA Information Survivability Conference and Exposition (DISCEX II)*, Anaheim, California, June 2001
- [2] W. Amme, N. Dalton, J. v. Ronne, and M. Franz, “SafeTSA: A Type Safe and Referentially Secure Mobile-Code Representation Based on Static Single Assignment Form”, in *Proceedings of the 2001 ACM Sigplan Conference on Programming Language Design and Implementation (PLDI 2001)*, Snowbird, Utah, June 2001.
- [3] Ch. H. Stork, V. Haldar, M. Beers, and M. Franz, *Tamper-Proof Annotations, By Construction*, Technical Report 02-10, Department of Information and Computer Science, University of California, Irvine, March 2002.
- [4] W. Amme, J. v. Ronne, and M. Franz, *Using the SafeTSA Representation to Boost the Performance of an Existing Java Virtual Machine*, Technical Report 06/02, Lehrstuhl Softwaretechnik, Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany, March 2002.