CS Report 01/2001
Proxy Compilation*

Matt Newsome and Des Watson †
{mattn,desw}@cogs.susx.ac.uk

January 2001

Abstract

In this paper, we outline new research concerning dynamic compilation of Java applications in environments where system resources are significantly limited. In such environments, which include “smart” mobile telephones and Personal Digital Assistants, memory and processor cycles can be scarce, making current techniques for the runtime translation of Java programs or program fragments inappropriate. We propose an alternative technique, *proxy compilation*, which makes use of idle, connected devices on a network to compile code on its behalf.

1 Introduction

The Java programming language[10, 21], although commonly associated with Internet programming, is a general-purpose object-oriented programming language. ■any of its features, such as the use of a garbage-collected memory allocation scheme, a virtual machine execution model and single class inheritance, have been highly lauded within the computer programming industry and academia.

The traditional role of the compiler[1, 13, 42, 23] has been to facilitate one-time translation of human-readable source programs into machine-readable object programs. The generated software is then order of magnitude. One common modification to this scheme, made in pursuit of faster program execution, has been to translate JV[redacted] code into native

2.1 Major components

The Java Virtual [redacted]achine JV[redacted]) is an abstract machine which processes JV[redacted] classfiles. Such classfiles the first reference to that symbol. The delay is generally in pursuit of increased execution speed: not all symbols in a classfile will be referenced during execution, so by delaying resolution, fewer symbols may need to be resolved with less runtime overhead. Additionally, the cost of resolution is

The JV[redacted] supports arrays as

Some innovative alternatives to the JV[redacted] classfile format

3.2 Proxy Compilation

One possible outcome of the research work described above could be that, owing to limitations on available time for the translation of classfiles at load-time, the runtime translation of JV[redacted] to LV[redacted] is impractical within a specific RO[redacted], RA[redacted] or stack limit above which, assumedly, the translation would be too intrusive or expensive). A sufficiently

4 Next Steps

We are currently implementing a dynamic compilation system to allow us to experiment with the ideas outlined above. The system is research-driven; consequently, we plan to support representative, but minimal Java programs. This implies supporting a subset of the Java 2 Platform API with, for example, complex I/O and networking facilities omitted. The AWT, Swing and JFC sections of the Java 2 Platform API will also be omitted, allowing us to focus on console-based applications.

Firstly, we are defining and implementing an initial LV■ in C++, together with a JV■ classfile JIT compiler for producing LV■ or target machine code at runtime. We expect to either implement a very simplistic garbage collector or use one of those freely available[2] under the GNU Public License[36]. We will consider the interaction of the LV■ with the garbage collector during this work; there may be some advantages to be gained from representing objects using a load/store V■

interpreter-based V■s for their high-speed startup times and small memory footprint, their argument is based upon their own Smalltalk V■. The relevance of their findings to Java systems is unclear, though assumedly a proxy compilation system benefits from the minimal V■ while also reaping the benefits of a powerful and flexible dynamic compilation system.

In the Ahead-of-Time domain, numerous static Java compilation systems exist[30, 9, 25, 24, 31, 11, 5, 38] though most are oriented to desktop systems with plentiful resources. All systems generate native binary code either directly or via ANSI C, which is subsequently compiled off-line. This approach is generally speed efficient, but often forbids use of dynamically loaded classes e.g. [4]) and suffers the overheads of native code relative to compact V■ bytecode.

The CO■POSE group's Harissa[25, 24] system notably acknowledges the requirement for statically-compiled Java applications to execute dynamically loaded classes, however, its solution - an interpreter - results in a slow, if compact solution. An equivalent interpreter has recently been added to the GNU gcj compiler[31]. Neither system addresses efficient runtime compilation of dynamically loaded Java code in resource-constrained environments.

Roelofs[32] notes the characteristics of resource-constrained systems, but is chiefly concerned with using connected devices to allow remote execution of application code. Our research instead seeks to use more powerful peers to speed translation of the device's core program for subsequent execution on the device itself.

Wakeman et al[26, 15] have worked on research which similarly acknowledges the problems of environments in which resources are limited. Their approach uses a proxy device to serve suitably compressed or scaled versions of requested data in accordance with client-specified constraints expressing, for example, degradation limits. This is analogous to the notion of proxy compilation, though the authors have not specifically proposed it. The work also proposes that clients inform the proxy of their resources. This is a potentially attractive technique which would allow the server to specialise a code fragment or application for the specific resources available to the client. In situations where low resources prohibit execution profiling, this may be the only feedback the client can provide regarding the runtime environment. An additional, albeit lesser, consideration is that their implementation uses Java and R■I on the client side. Our work directly addresses the question of proxy compilation and is designed to scale to very simple clients where a Java runtime environment may not be feasible.

The vast majority of dynamic compilation systems require storage of a dynamic compiler system in the runtime environment, and must execute on the target system. The small number of projects which do not employ this model are now described. Voss and Eigenmann[41] detail a system which is notionally similar to proxy compilation, but assumes various system characteristics. These include a requirement for NFS mounted storage to be shared between systems and a reliance on RPC facilities. We believe such a solution would not scale well to resource-constrained systems particularly single threaded applications which use a minimal operating system or do not require an OS). Additionally, this project has focussed on ANSI C and FORTRAN applications rather than Java.

Bell Labs' Inferno system[44, 43, 7] and Tao Systems' Elate/Intent system[12] both use a low-level V■ instruction set to increase the efficiency of Java code. These two systems are now contrasted with our proposed systems.

Inferno's use of a memory-to-memory virtual machine results in a virtual machine architecture which is superficially similar to our proposed LV■ system. There are a number of critical differences, however. Firstly, Inferno is target-independent, supporting Intel x86, SPARC, AR■, PowerPC, ■IPS and other devices. Although the principle of a low-level virtual machine is applicable to targets with a load/store architecture, we expect to increase efficiency by creating specialised versions of the LV■ instruction set for individual processors. Furthermore the Inferno virtual machine Dis) has an instruction set which has been designed for the Limbo programming language, not Java. Although there are many similar features, including objects and garbage collection, supported by an[c)1999.ec devices.)Tj3ext99

Elate/Insight also uses a low-level, target-independent instruction set, however it, like the Kimera project[35], use a form of remote compilation which relies on shared memory and persistent network connections. Such systems fail to acknowledge the often intermittent nature of network connections to resource-constrained devices. As described above, our proxy compilation scheme is designed to scale to a

[4] Natural Bridge. Bullet-Train Homepage.
`http://www.naturalbridge.com/bullettrain.html`.

[5] Department of Computer Science and Engineering, University of Washington.
Cecil/Vortex Homepage.
`http://www.cs.washington.edu/research/projects/cecil/www/index.html`.

[6] Stephan Diehl. A Formal Introduction to the Compilation of Java. *Software – Practice and Experience*, 28 3):297–327, March 1998.

[7] Sean Dorward, Rob Pike, David Leo Presotto, Dennis Ritchie, Howard Trickey, and Phil Winterbottom. Inferno. In *Proceedings of the IEEE Compcon 97 Conference*, pages 241–244, San Jose, 1997.

[8] Ericsson Mobile Communications AB. Official Bluetooth website.
`http://www.bluetooth.com`, 1999.

[9] Robert Fitzgerald, Todd B. Knoblock, Erik Ruf, Bjarne Steensgaard, and David Tarditi. Marmot: An Optimizing Compiler for Java. Technical Report MSR-TR-99-33, Microsoft Research, June 1999.

[10] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java^{TM}Language Specification.* Addison-Wesley, 2nd edition, 1999.

[11] Silicomp Group. Turbo-JHomepage.
`http://www.ri.silicomp.fr/adv-dvt/java/turbo/`.

[12] Tao Group. Elate/Insight Homepage.
`http://www.tao.co.uk`.

[13] Dick Grune, Henri E. Bal, Ceriel J.H. Jacobs, and Koen G. Langendoen. *Modern Compiler Design.* John Wiley and Sons, Ltd., 2000.

[14] Joseph Hummel, Ana Azevedo, David Kolson, and Alexandru Nicolau. Annotating the Java Bytecode in Support of Optimization. *Concurrency: Practice and Experience*, 9 11):1003–1016, November 1997.

[15] Ian Wakeman, Andy Ormsby, and Malcolm McIlhagga, School of Cognitive and Computing Sciences, University of Sussex. An Architecture for Adaptive Retrieval of Networked Information Resources. *IEE Colloquium on*

[22] Blair [redacted]cGlashan and Andy Bower, Object Arts Ltd. The Interpreter is Dead Slow). Isn't it? Position Paper for OOPSLA'99 Workshop: Simplicity, Performance and Portability in Virtual [redacted]achine design.
http://www.squeak.org/oopsla99_vmworkshop/, October 1999.

[23] Steven S. [redacted]uchnick. *Advanced Compiler Design and Implementation.* [redacted]organ Kaufmann, 1997.

[24] G. [redacted]uller, B. [redacted]oura, F. Bellard, and C. Consel, IRISA / INRIA, University of Rennes. Harissa: A Flexible and Efficient Java Environment [redacted]ixing Bytecode and Compiled Code. In *3rd Usenix*

[41] [redacted]ichael J. Voss and Rudolf Eigenmann. A Framework for Remote Dynamic Program Optimization. In Jong-Deok Choi, editor, *Proceedings of the ACM SIGPLAN Workshop on Dynamic and Adaptive Compilation and Optimization (Dynamo '00)*