An formal specification of the Java Virtual Machine and its verification

Shigeru OKUMURA

Language Design Labratory
School of Information Science,
Japan Advanced Institute of Science and Technology

February 15, 2000

Keywords: Java Language, Java Virtual Machine, Bytecode Verifier, Specification, CafeOBJ.

1 Introduction

Java Language security model is base on bytecode verifier, (applet)class loader, and Java security manager. The most important section in this security model is bytecode verifier. but, the specification of bytecode verifier, also the specification of Java virtual machine, has been written in natural language instead of formal language. If bytecode verifier were incorrectly imprementation, it is a serious problem for Java security model.

In this paper, we check in detail the specification of Java bytecode verifier written in natural language and described its specification in CafeOBJ (algebraic specification language).

2 CafeOBJ

CafeOBJ is a multi-paradigm algebraic specification language which is a successor of OBJ. CafeOBJ is based on the combination of several logics consisting of many softed algebra, order softed algebra, hidden algebra and rewriting logic.

According to its semanics, CafeOBJ can fit in several specification and programming pradigms such as equational specifications and programmings, reqriting logic specification, behavioural concuretn specifications. Object orientation is a derived feature of CafeOBJ which can be treated both in behavioural specification and rewriting logic; in this paper we consider only the behavioural specification approach. CafeOBJ has a powerfull module

system: several kinds of module imports, parameterized modules, and for each module on can choose between loose and tight semantics.

CafeOBJ is executable which means it can be used for rapid prototyping and theorem proving. Its operational semantics is based on term rewriting, and the proof calculi are equational and rewriting logic proof calculi. For confluent and terminating specification, two terms are equal when their normal forms are identical.

3 Security

Security is extremely important in Java. If people are going to download third-party programs and run them on their computers, they need to be certain that the third-party program won't cause any damage. For instance, downloaded applets must not:

- damage the computer hardware or file system.
- cause the computer to crash or go so sluggishly that it becomes unusable.
- transmit details about the computer or contents of its files to third parties.

If an applet could do any of these things, people would be unwiling to run applets downloaded from an external source.

Java employs a number of techniques to manage security. For example, calls for system resources, such as sockets and files, are monitored by Java. Java runtime systems can trap many of the operations that ann applet or an application wants to perfrom.

4 Java Virtual Machine and Bytecode Verifier

Java Virtual machine usually loads class file, and unfold on the memory in Java Virtual machine. This class file includes executable code, and this code are interpreted on Java Virtual machine. While interpreting code,the thread, a unit of execution, has Java Stack on the memory, and the Java Stack often has the frame. The frame are stacked on the Java Stack when Java Virtual machine call method, and the frame has the operand stack and the register. While executing the method, Java Virtual Machine are using operand stack and the register.

In most case, the Java class files created by Java byte compiler from Java source code, and excute on Java virtual machine. Java class file can download class file from remote machine or local file system, and then excute on local machine. Before excution of class file on Java virtual machine, the virtual machine is checking class file because of safety excution. If Java class file is broken or altered, Java virtual machine excution without checking class file tend to wrong state.

Whenever Java downloads a class file from a remote site, it "verifiers" the class file. Verification is mostly concerned with ensuring that loading and running the code in the class file will not crash the Java Virtual Machine, leave the interpreter in an undefined state, or crash the host computer. Verification involves applying a number of "structural

constraints" to the class file. For example, the verifier checks the layout of the data in the class file, making sure that:

- The data format of the class file is correct.
- All type descriptors are well-formed and consistent.
- All named fields, methods and classes exist.
- More importantly, the verifier performs "bytecode verification" using the Java bytecode verifier.

The Bytecode verifier is a sophisticated program that checks the bytecode within a method to ensure that it is well-behaved.

The bytecode verifier performs a range of basic checks. For Example, it checks that:

- All goto and branch instructions refer to a valid bytecode address
- The types of all instruction parameters are corrent.

In addition, the class file verifier runs a "theorem prover" on the virual machine code in each method. The theorem prover checks the code at a much deeper level. After running the theorem prover, the verifier knows that:

1. The opearnd stack is used consistently and doesn't overflow or underflow.
2. The local variables are used consistently and correctly.
3. Methods are called with the right number and types of arguments.

Concluding these things and the specification of bytecode verifier written in natural language, we developed the model of the bytecode verifier and described the specification of the bytecode verifier in CafeOBJ. The meaning of describing the formal specification of bytecode verifier is that it can check whether its sepecification has vague thing or not, and also can check the bytecode verifier's guarantee do not break.

5 Conclusion

We described the specification of the Bytecode verifier based on the specification of written in natural language. Because the specification is written in CafeOBJ, this specification can be executable on sample examples. But, we do not described the Java Virtual machine yet. We should describe it and we should verify that the code don't go wrong execution on the Java virtual machine while the code of passing the Java bytecode verifier check.