Implementing Multiple Protection Domains in Java

Chris Hawblitzel Chi-Chao Chang, Grezgorz Czajkowski
Deyu Hu, and Thorsten von Eicken

Presented by Wei Li

1 Introduction

Traditional operating systems use virtual memory to enforce protection between processes. In the past decade of operating systems research, a large number of fast inter-process communication mechanisms have been proposed. Nevertheless, the cost of passing through the kernel and of switching address spaces remains orders of magnitude larger than that of calling a procedure.

The robustness versus performance tradeoff is persuasive in component software. This paper explores the use of safe language technology to offer high performance as well as protection in a software component environment. In a safe language, calls across protection boundaries could potentially be as cheap as simple function calls. We assume that the protection mechanism should enforce module or class structures. Thus communication should only be possible through well-defined interfaces, and not through side effects.

Several projects have recently described how to build protection domains around component in a safe language environment. All of these approaches have the following problem: using objects references as capabilities leads to severe problems with revocation, resource management, and inter-domain dependency analysis. To overcome the limitations of the straight-forward approach, we implemented J-Kernel, providing sophisticated capability-based protection features.

2 Language-based protection background

The term namespace can be used to express the restriction for memory access: a name space is a partial function mapping names of operations to the actions taken when the operations are executed. Protection domains around software components can be constructed in a safe language system by providing a separate namespace for each component. Java provides three basic mechanism for controlling namespaces:

- Selective sharing of object references
- Static access control
• Selective class sharing

The simple controls over the namespace in Java can be used to construct software components that communicate with each other but are still protected from one another. This simple share anything approach seemed the natural basis for a protection system. However, as we worked with this approach, a number of problems became apparent.

• Revocation
• Inter-domain Dependencies and Side Effects
• Domain Termination
• Threads
• Resource Accounting

3 The J-Kernel

The J-Kernel is a capability-based system that supports multiple, cooperating protecting domains which run inside a single Java virtual machine.

The primary goals of the J-Kernel are:
• a precise definition of protection domains, with a clear distinction between objects local to a domain and capability objects that can be shared between domains
• well defined, flexible communication channels between domains based on capabilities
• support for revocation for capabilities
• clean semantics of domain termination

Protection in the J-Kernel is based on three core concepts – capabilities, protection domains, and cross domain calls:

• Capabilities are implemented as objects of the class Capability and represent handles onto resources in other domains
• Protection domains are represented by the Java class Domain. Each protection domain has a namespace that it controls as well as a set of threads
• Cross-domain calls calls are performed by invoking methods of capabilities obtained from other domains.

The J-Kernel’s implementation of capabilities and cross-domain calls relies heavily on Java’s interface classes and remote method invocation (RMI) specification.

Internally, RMI calls will automatically generate a stub class at run-time for each target class. This avoids off-line stub generators and IDL files, and it allows the J-Kernel to specialize the stubs to invoke the target methods with minimal overhead. Besides switching domains, stubs have three roles:
• copying arguments
• supporting revocations
• protecting threads

J-Kernel domains are given considerable control over their own class loading. Each domain has its own class namespace that maps names to classes. A domain’s namespace is controlled by a user-defined resolver, which is queried by the J-Kernel whenever a new class name is encountered.

Shared classes are the basis for cross-domain communication: domains must share remote interfaces and fast copy classes to establish common methods and argument types for cross-domain calls.

4 An Extensible Http Server

One of the applications for J-Kernel is an extensible HTTP server. The goal is to allow users to dynamically extend the functionality of the server by uploading Java Programs, called servlets.

The J-Kernel runs within the same process IIS and includes a system servlet with access to native methods that allows it to receive HTTP requests from IIS and return corresponding replies.

The measurements show that ISAPI bridge accounts for half of the degradation and only the remainder is directly attributed to J-Kernel.

5 Related Work

• JDK 1.1’s authentication
• JDK 1.2’s protection domain
• SPIN project
• Typed assembly language
• Very fast inter-process communications

6 Conclusion

This paper explores the use of safe language technology to construct robust protection domains. The advantages of using language-enforced protection are portability and good cross-domain performance. The most straightforward implementation is to use object references directly as capabilities. We
discussed all the problems and how to solve the problem with the implementation of J-Kernel. And we argue to use a structured approach: only capabilities can be shared, and non-capability objects are confined to single domain.