Strong Protection Domains and Resource Control With Java™ Technology

Walter Binder
Software Engineer
CoCo Software Engineering GmbH
Learn about the requirements of component execution platforms and see how such systems can be implemented entirely with Java™ technology.
Learning Objectives

• As a result of this presentation, you will be able to understand:
  – Why Java™ technology-based components need protection
  – How protection domains can be efficiently implemented entirely with Java technology
  – The design of an efficient communication model
  – How to ensure safe domain termination
  – Techniques for resource control in the Java programming language
Speaker’s Qualifications

• Developer using Java technology since 1996
• Research on Java technology-based operating systems since 1997
• Designer and developer of J-SEAL2
• Working on resource control for the Java platform
Agenda: Java™ Component Execution Platforms

• Java™ component execution platforms
• Strong protection domains
• Safe domain termination
• Efficient and mediated communication
• Resource control
Java Components

- Applets
- Servlets
- Enterprise JavaBeans™
- Mobile objects (mobile agents)
Requirements for Component Execution Platforms

• Security
  – Protect platform and components
  – Ensure secrecy
  – Ensure integrity
  – Prevent denial-of-service attacks
  – Prevent resource leaks

• Portability

• High performance
Why Java™ Technology for Components?

• Portable code
• Language safety
  – Type safety
  – Automatic memory management
  – Memory protection
  – Byte-code verification
• Multi-threading
• Class-loader namespaces
Caveats and Pitfalls

• The JVM™ virtual machine is not an operating system
• No protection domains
• Uncontrolled aliasing
• No ownership information in objects
• Covert channels (e.g., public static variables)
• No resource control
• Single heap
Possible Solutions

• Separate JVM for each component
  – Requires process support in operating system
  – High startup overhead, reloading of JDK™ software-based classes (“JDK classes”)
  – Expensive communication
  – Limited scalability

• Multiple protection domains in single JVM
  – Difficult to implement
Separate JVM™ for Each Component

JVM 1
Component 1

JVM 2
Component 2

JVM 3
Component 3
Multiple Protection Domains in Single JVM™
Design Goals

• Operating system structure
  – Isolated protection domains
  – Safe domain termination
  – Mediated communication
  – Resource control
• Micro-kernel architecture
• Based entirely on Java™ technology
• Efficient implementation techniques
Case Study: The J-SEAL2 Mobile Agent Kernel

- Strong security guarantees
- Formal model: Seal Calculus (Jan Vitek)
- Redesign of the “JavaSeal” project (University of Geneva)
- Small micro-kernel (< 150KB)
- Portable (based entirely on Java technology)
- Flexible and extensible
- Efficient and scalable
Related Work

• Portable (based entirely on Java technology)
  – Project “JavaSeal”
  – J-Kernel
  – (JRes)

• Non-portable (native code, modified JVM™)
  – Alta
  – KaffeOS
  – Luna
  – Nomads
Agenda: Strong Protection Domains

- Java™ component execution platforms
- Strong protection domains
- Safe domain termination
- Efficient and mediated communication
- Resource control
Requirements

• Isolation of components
• Kernel enforces domain boundaries
• Encapsulation
  – Class-loader
  – Classes
  – Threads
  – Objects
  – Kernel state (e.g., communication queues)
Class-loading

• Shared classes
  – Loaded by system class-loader
  – JDK™ and kernel classes

• Replicated classes
  – Loaded by class-loader of protection domain
  – Libraries and component classes

• Caching of replicated library classes
Extended Byte-code Verification

- Limit access to JDK™ classes and kernel
- Configurable verifier directives
  - Restricted access to packages and classes
  - Restricted access to class members
  - Extension restrictions
- Efficient constant-pool verification
Case Study: Nested Protection Domains in J-SEAL2

- Hierarchy of protection domains
- Parent domain controls children
  - Communication control
  - Resource control
  - Termination of sub-hierarchies
- Sandboxes with different privileges
J-SEAL2 System Structure

Services:
- Net
- GUI
- Mail

Stationary Domains:
- Sandbox
  - Trusted
  - Anonym

Mobile Agents:
- Agent 1
- Agent 2
- Agent 3
- Agent 4
Agenda: Safe Domain Termination

- Java™ component execution platforms
- Strong protection domains
- Safe domain termination
- Efficient and mediated communication
- Resource control
Requirements

• Immediate resource reclamation
  – Termination of all threads in domain
  – Invalidation of open communication ports

• Ensure consistency of JVM™ and kernel
  – Atomic kernel operations
  – VM operations in kernel (e.g., class-loading)
Code Restrictions

- No finalizers
  - void finalize()
  - void classFinalize()
- No catching of ThreadDeath
- Kernel enforces restrictions
  - Byte-code rewriting
  - Extended byte-code verification
Kernel Mode

• Atomic operations

• Multiple-reader/single-writer lock
  – Domain termination: Write lock (exclusive)
  – Other kernel operations: Read lock (shared)

• Asynchronous class-loading
  – Requires read lock
  – Prevent deadlocks
  – Keep track of threads in kernel-mode
boolean excl = ...; // exclusive or shared
Kernel.lock(excl);
// now executing in kernel-mode
try {
    // allocate all objects (worst case)
    ...
    // update kernel state
    ...
} finally {
    Kernel.unlock(excl);
}
// now back in user-mode
Agenda: Efficient and Mediated Communication

- Java™ component execution platforms
- Strong protection domains
- Safe domain termination
- Efficient and mediated communication
- Resource control
Communication Models

• Direct sharing
  – Java™ technology based references (“Java references”)
  – Special remote pointers (e.g., Luna)

• Indirect sharing (e.g., J-SEAL2, J-Kernel)

• Copying (e.g., project “JavaSeal”, J-SEAL2, J-Kernel)
Direct Sharing With Java™ Technology

JVM

Domain 1

Domain 2
Direct Sharing With Java™ Technology: Pros and Cons

• Pros
  – Simple
  – Fast (method invocation)
  – Based entirely on Java technology

• Cons
  – Blurs domain boundaries
  – No revocation
  – Domain termination
  – Resource accounting
Direct Sharing in Luna
Direct Sharing in Luna: Pros and Cons

- Pros
  - Fast (method invocation)
  - Revocation
  - Domain termination
  - Resource accounting

- Cons
  - Requires modified JVM™
Indirect Sharing
Indirect Sharing: Pros and Cons

• Pros
  – Revocation
  – Domain termination
  – Resource accounting
  – Based entirely on Java technology

• Cons
  – Overhead (indirection)
Copyring
Copying: Pros and Cons

• Pros
  – Revocation not necessary
  – Domain termination
  – Resource accounting
  – Based entirely on Java technology

• Cons
  – High overhead for large object graphs (memory and CPU)
  – Serialization restrictions
Case Study: Communication in J-SEAL2

- **Channels (copying)**
  - Name, location (self, parent, child name)
  - Only between neighbor domains
  - Synchronous send/receive (Seal Calculus)
  - Optional timeouts
  - Asynchronous send

- **External references (indirect sharing)**
  - Cross-domain method invocation

- **Inter Agent Method Calling (IAMC)**
Communication Values

- Copying and indirect sharing
- Capsule: Serialized object graph
- Special treatment of external references
- Copying optimizations
  - Primitive types
  - Arrays of primitive types
  - Strings
External References

Shared object
Descendant

Parent P
Rp

Child C
Rc
SO

Child D
Rd
Agenda: Resource Control

- Java component execution platforms
- Strong protection domains
- Safe domain termination
- Efficient and mediated communication
- Resource control
Requirements

- Prevention against denial-of-service attacks
- Resource control
  - Resource accounting
  - Enforcing restrictions
- Physical resources (e.g., CPU, memory, net)
- Logical resources (e.g., threads, domains)
Implementation

- Accountability (objects belong to 1 domain)
- Byte-code rewriting
  - CPU: Count byte-code instructions
  - Memory: Check before allocation (JRes)
- Compensation for native code
  - Class-loading
  - Deserialization
  - Reflection
  - Object cloning
Memory Account (Simplified)

- Memory limit for multiple threads
- Synchronization
- Weak references to allocated objects

```java
public final class MemAccount {
    public void setLimit(int limit);
    public void checkAllocation(int size)
        throws ResourceOveruseException;
    public void register(Object o);
}
```
CPU Account

• Separate account for each thread
• No synchronization
• Periodic scheduler thread (high priority)

```java
public final class CPUAccount {
    public volatile int usage;
}
```
Passing Accounting Objects

• Unmodified method

```java
Object f(int x) {
    if (x < 0) return null;
    else return new Foo(g(x));
}
```

• Added accounting objects

```java
Object f(int x, MemAccount mem, CPUAccount cpu) {
    if (x < 0) return null;
    else return new Foo(g(x, mem, cpu), mem, cpu);
}
```
Rewriting for Memory Accounting (Simplified)

```java
Object f(int x, MemAccount mem, CPUAccount cpu) {
    if (x < 0) return null;
    else {
        int y = g(x, mem, cpu);
        mem.checkAllocation(SIZEOF_FOO);
        Object o = new Foo(y, mem, cpu);
        mem.register(o);
        return o;
    }
}
```
Rewriting for CPU Accounting

```java
Object f(int x, MemAccount mem, CPUAccount cpu) {
    cpu.usage += 8;
    if (x < 0) {
        cpu.usage += 8;
        return null;
    }
    else {
        cpu.usage += 26;
        int y = g(x, mem, cpu);
        mem.checkAllocation(SIZEOF_FOO);
        Object o = new Foo(y, mem, cpu);
        mem.register(o);
        return o;
    }
}
```
Overhead of CPU Accounting

The graph shows the relative execution time of various applications with different Java technologies. The applications include:

- mtrt
- jess
- compr.
- db
- mpeg
- jack
- javac
- geo. mean

The graph compares the overhead of CPU accounting using:

- Ubench-Ujdk
- Rbench-Ujdk
- Rbench-Rjdk

The x-axis represents different benchmarks, and the y-axis represents the relative execution time.
Case Study: Hierarchical Resource Control in J-SEAL2

- **Startup**
  - Root domain owns all resources

- **Subdomain creation**
  - Parent may donate resources to child
  - Parent may share resources with child
Resource Donation and Sharing in J-SEAL2

Trusted Domains

Root

CPU

∞

MEM

∞

Untrusted Applications

CPU

5%

MEM

40MB

CPU

15%

MEM

10MB

Root

CPU

0%

MEM

∞

share

split 20%

share

split 50MB

share

split 10MB

share

split 75%

share

split 15%

share

split 10MB
Summary

• Component execution platform requirements
  – Security
  – Portability
  – High performance and scalability

• Operating system functionality for the Java™ platform
  – Protection domains
  – Safe domain termination
  – Mediated communication
  – Resource control
Get More Information!

• Email to w.binder@coco.co.at
  – Articles
  – Thesis
  – Background information
  – Specific questions
  – Collaboration
  – Research and evaluation versions of J-SEAL2
Q&A
Question

- Which restrictions does the protection model impose on components?
Answer

• ‘Dangerous’ JDK™ software functionality
• Restructure applications for security
• Use of service components
• Automatic transformation of components