# Creating Manageable Systems With JMX

**Vladimir Vivien**
Simplius, LLC

vvivien@simpli.us
http://simpli.us/

# Goal

Explore how to incorporate runtime manageability into your systems using Java™ Management Extensions (JMX™).

# Agenda

- Motivation
- Introduction to JMX Technology
- Hello MBean
- Tooling JMX
- Implementing Manageability
- Summary
- Q&A

# Motivation

# Are We There Yet?

- Would you drive this car?
  - No speedometer
  - No gas gauge
  - No engine temp gauge
  - No instrument panel
  - No signal flashers
- Most deployed apps have similar shortcomings
  - System is a black box
  - No runtime visibility
  - No instrumentation panel

# Motivation

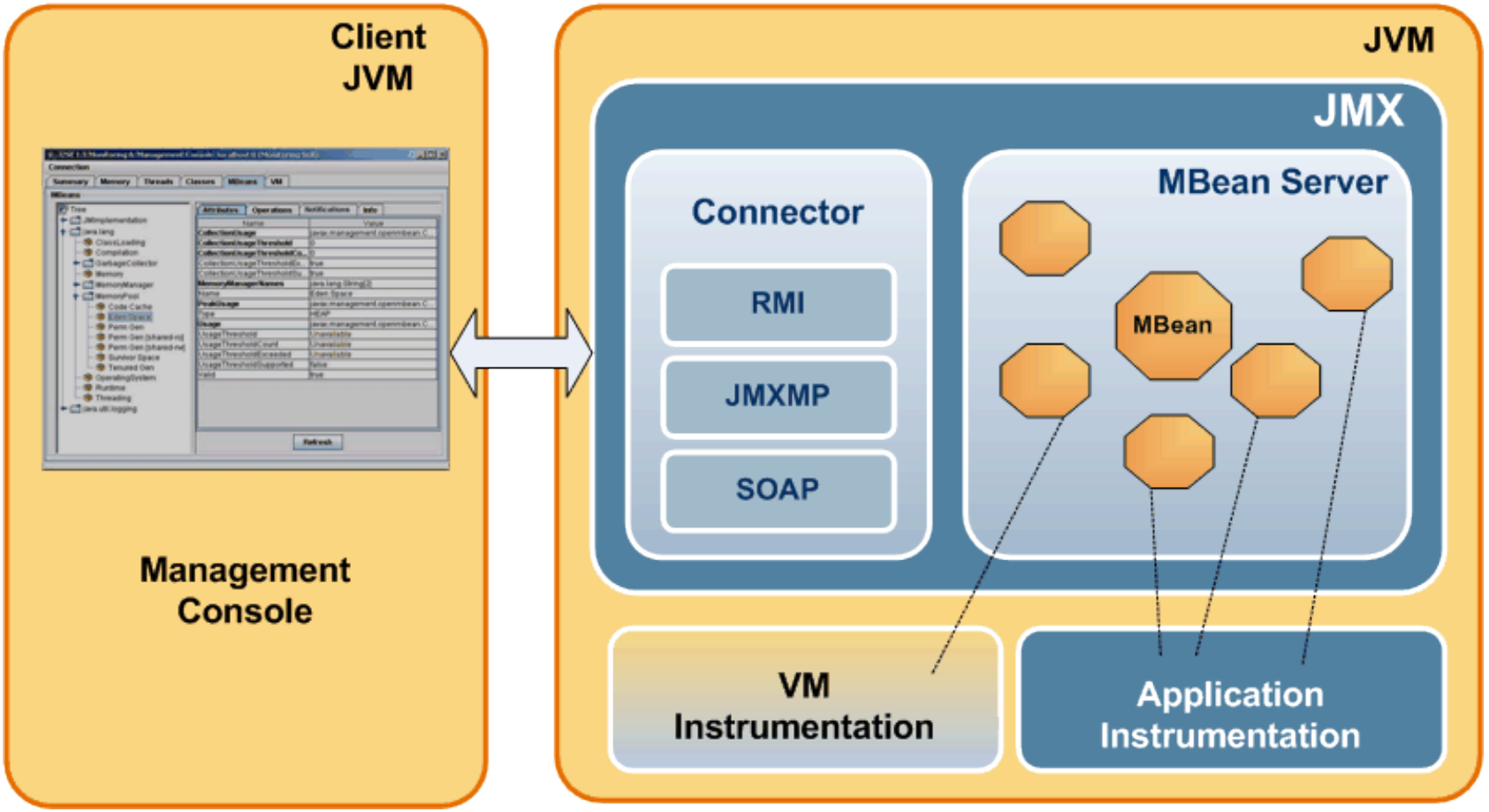
- Better visibility of system at runtime
- Go beyond simple event log files
- Ability to monitor
  - Expose states/health of application in real time
  - Take preventive measures where possible
  - React intuitively and quickly to changes & requirements
- Greater management
  - Interactively control application in user-friendly way
  - Ability to change operational parameters on the fly
  - Avoid down time for critical applications

# Introduction to JMX Technology

# What Is JMX?

- Java Management eXtension
- Standard API for runtime management & monitoring
- Mature API [originally (JSR)-3]
  - Includes a server, specialized beans, wire protocols and console
  - Component-based development
  - Provides full event model
- Part of core Java platform starting with version 5
  - Runtime VM information instrumented
  - Instrument and expose your own application

# JMX Architecture



# MBean Server

- Container for management components
  - Allows registry, discovery, and query of components
  - Handles interaction with components from client
  - Exposes component operations and properties
- Provides event bus
  - Register to receive event notification
  - Broadcast events to registered components
- Server is exposed as management component

# The Management Bean (MBean)

- Exposes instrumented data and control
- MBean API:
  - *Standard MBean* interface
    - The simplest way to implement management
    - Requires naming pattern (i.e. xxxxMBean)
  - Others include *Dynamic*, *Model*, and *Open MBean*
  - Registered in the MBean Server
  - Exposes *attributes* (JavaBeans™ getters/setters)
  - Expose *operations* (interface methods)

# MXBean

- New type of MBean (since Java 5)
- Provides same functionalities as MBean
- Allows exposure of complex data type
- Client has no dependency on type
- Data available as a map
- Supports type relationship
- Supported by JConsole

# VM Instrumentation

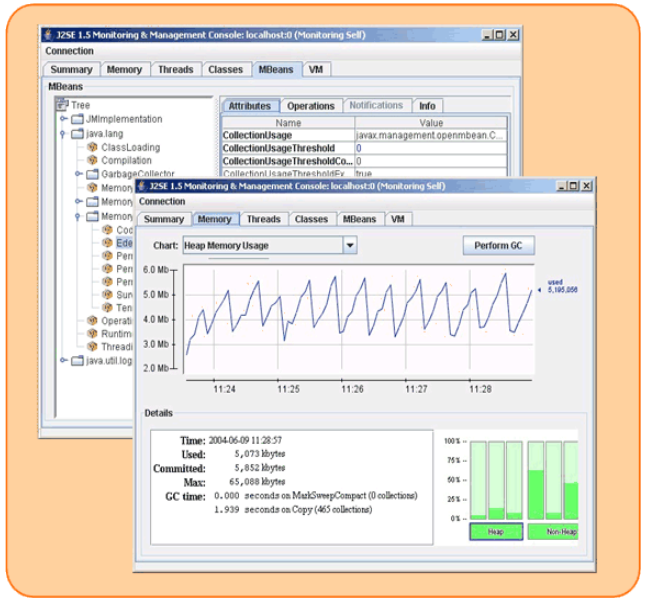
- Starting with Java platform v.5, VM includes instrumentation
- VM exposes several platform MXBeans
  - Class loading information
  - Memory pool / consumption
  - Garbage collection
  - Operating system
  - Threads
  - And more...
- Ability to look into VM's activities with no code

# Connector Architecture

- Allows remote management
- Client uses MBeans as if they are local
- JMX provides default RMI connector
- Supports JRMP and ORB-IIOP
- Connector can be secured
- Simple API
  - JMXServiceURL
  - JMXConnectorServer
  - JMXConnector

# Management Console - JConsole

- Interactive desktop management tool
- Introduced in Java Development Kit (JDK™) 5 release
- Profiles memory, threads, GC
- Supports for remote connection
- Attaches to running VM
  - Java platform v.5 requires switch
  - `-Dcom.sun.management.jmxremote`



# Remote Access Security

- Secured access to remote MBean server
- Supports several security methods
  - Password / access files
    - `{JRE_HOME}/lib/management`
  - SSL Certificate
  - JAAS
- Uses TLS communication
- Control of remote security (system properties)
  - `com.sun.management.jmxremote.authenticate=false`
  - `com.sun.management.jmxremote.ssl=false`

# Demo

# Hello MBean

# Instrumenting Your Code

- Decide on attributes & operations to expose
- Write management interface
- Implement management class
- Register management with MBean Server

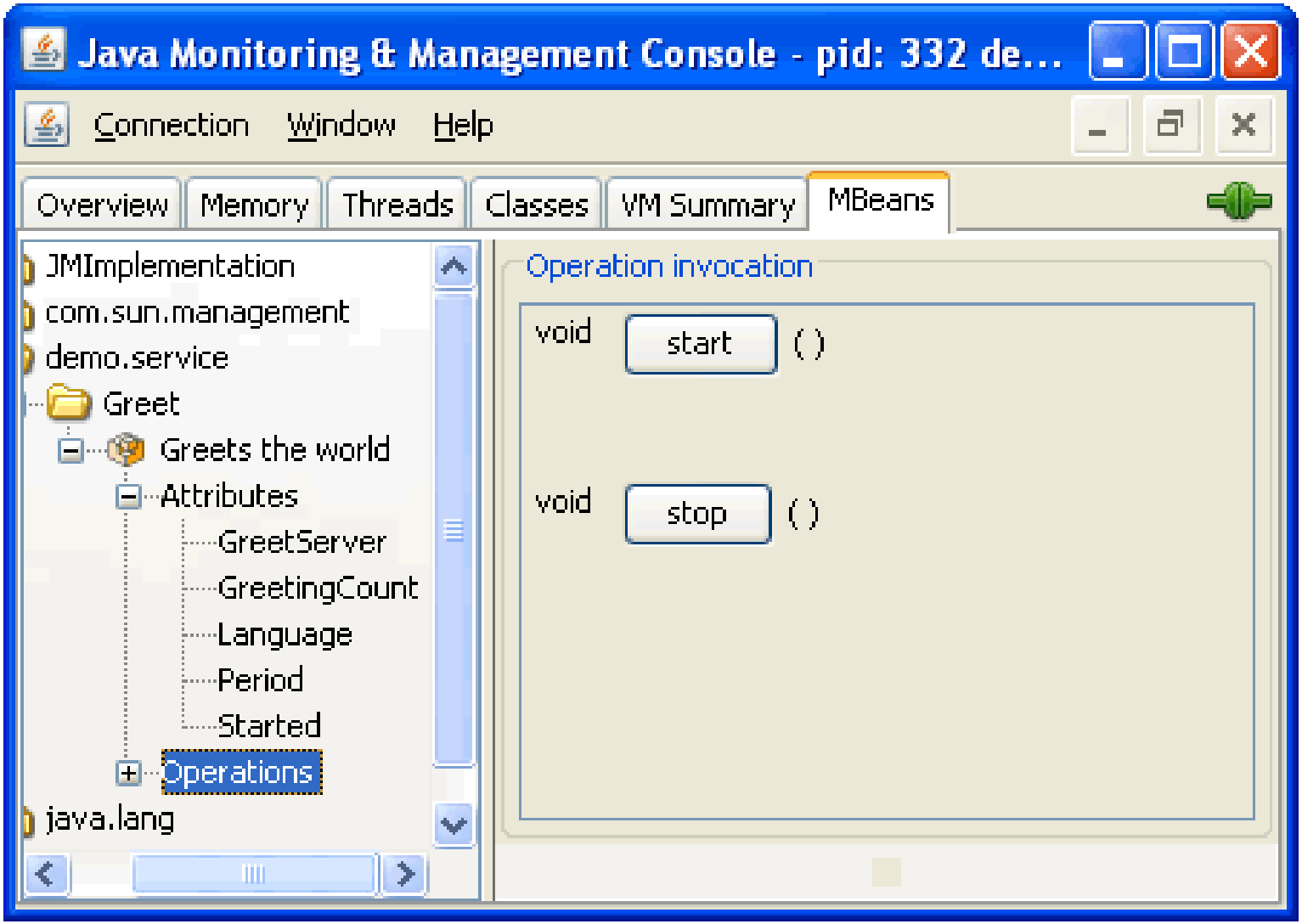
# Exposing an MBean – Code

1
```
public interface GreetServiceMBean {
  public void setLanguage(String lang);
  public void start();
  public void stop(int val);
}
```

2
```
public class GreetService implements GreetServiceMBean {
  public void setLanguage(String lang){...}
  public boolean start() {...}
  public void stop(int val) {...}
}
```

3
```
public void RegisterMBean () throws Exception {
  MbeanServer svr = ManagementFactory.getPlatformMBeanServer();
  ObjectName objName =
    new ObjectName("demo.service:type=Greet...");
  svr.registerMBean(new GreetService(), objName);
}
```

# Exposing an MBean – Console View



# The Object Name

- `javax.management.ObjectName`
- Uniquely identifies registered MBeans
- Supports flexible naming strategy
- Format
  - **`[domain]:key=value[,key=value]*`**
- Example
  - *demo.service:type=Greeting,description=greets the world*
- Use by management tools extensively

# JMX Event Notification Model

- JMX technology has a rich notification model
- Supports asynchronous component communication
- Interfaces included in JMX technology notification model
  - Notification - content of notification
  - NotificationBroadcaster – source of event notification
  - NotificationListener – recipient of notification
  - NotificationFilter – Allows filtering of notifications
- The MBean server broadcasts numerous events
- Your Mbean components can broadcast/listen for events
- Management tools can subscribe to notifications

# Listening for Event Notifications

1

```
public class TimerLstnr implements NotificationListener{
  public handleNotification(Notification n, Object obj){...}
}
```

2

```
public class RegisterListener throws Exception{
  MbeanServer server = ManagementFactory.getPlatformServer();

  javax.management.timer.Timer timer = new javax...Timer();
  ObjectName broadcaster = new ObjectName("demo:svc=Timer");
  timer.addNotification("timer.heartbeat",
      null, null, new Date(System.currentMillis()), 3000L);
  server.registerMBean(timer, broadcaster);
  timer.start();

  ObjectName listener = new ObjectName("demo:obj=TimerLstnr")
  NotifycationFilterSupport f = new NotificationFilterS...
  f.enableType("timer.heartbeat");
  server.addNotificationListener(
      broadcaster, listener, f, null);
}
```

# Demo

# Tooling JMX

# Useful Tools for JMX

- Programmatic API's
  - **Spring JMX**
  - **Groovy MBean**
  - **AOP (with Spring)**
  - JBoss Service Components
  - Geronimo GBean
  - Commons Modeler
- Consoles
  - MC4J
  - Glassbox
  - JManage

# JMX and Spring

- Declaratively registers any POJO as MBeans
- No special interface or naming patterns required
- Full control and flexibility:
  - Strategies for locating available MBean Server
  - Provides MBean Exporter
  - Support several strategies for ObjectName
  - MBeanInfoAssembler API
    - **Use code-level annotation to customize export**
    - **Auto-detect MBeans** based on naming pattern
    - Use **list of interfaces** for management
    - Use **Method names** to export for management
  - Specify MBean registration behavior

# JMX and Spring (Cont.)

- Wire POJO's as JMX notification listeners
- Support for JMX MBean proxies
- Support for JMX remoting (JSR 160)
- Use Spring's supplied MBean server or provide your own (defaults to VM's)
- Spring augments JMX as DI container
  - Declarative injection of MBean dependencies
  - Easily establish relationship between MBeans

# JMX with the Spring Framework

1

```
@ManagedResource(objectName="demo.service:type=Greet ...")
public class GreetService {
  @ManagedAttribute
  public void setLanguage(String lang){...}
  ...
}
```

2

```
<bean id="greetSvc" class="demo.GreetService"/>
<bean id="greetingServer" class="demo.service.GreetingService"/>

<bean
  id="attribSrc"
  class="...export.annotation.AnnotationJmxAttributeSource"/>

<bean id="exporter" class="...export.MBeanExporter">
  <property name="autodetect" value="true"/>
  <property name=”assembler”>
    <bean class="...export.assembler.MetadataMBeanInfoAssembler">
    <property name="attributeSource" ref="attribSrc"/>
    </bean>
  </property>
</bean>
```

# A Groovier JMX

- Popular scripting language and described as
  - "***Agile dynamic language*** *for the Java platform* ***inspired*** *by* ***Python***, ***Ruby***, *and* ***Smalltalk***..."
- Compiles directly into Java bytecode
  - Groovy scripts import Java objects (vice versa)
- Built-in support for JMX
- Provides scriptability for JMX artifacts
- Access JMX MBean values and operations
- **Integrates well with the Spring Framework**

# Monitoring with Groovy

```
def objName = "demo.service:type=GreetingServer"
der server  = ManagementFactory.platformMBeanServer
def bean    = new GroovyMBean(server, objectName)
println "Language $bean.language"
```

# Demo

# Implementing Manageability

# School of Management

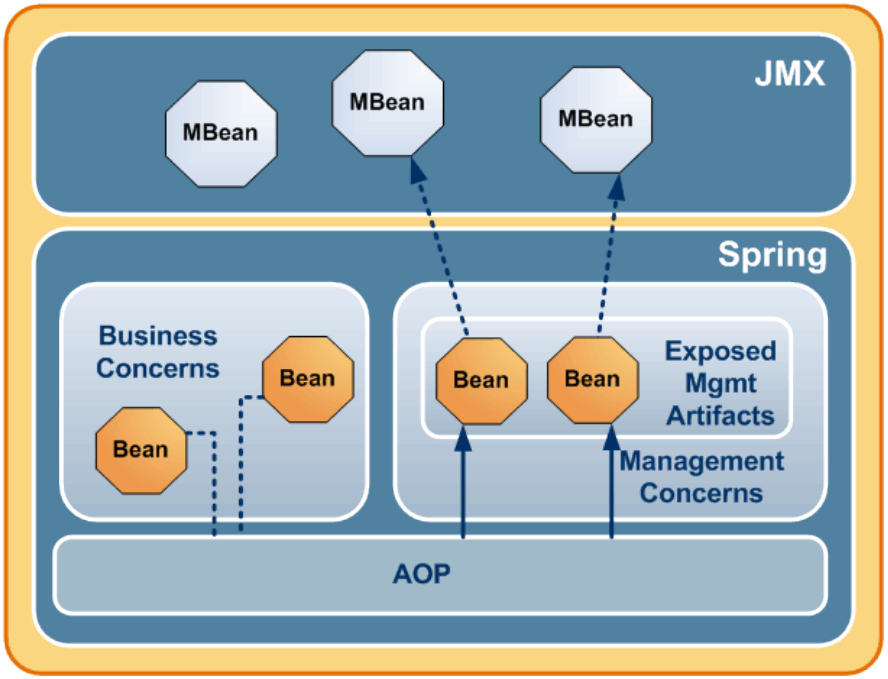
- Separate management of system from functionality of system
- When to Manage
  - Faceless systems
  - Long running processes
  - Services
  - Access system information & control
- Types of Management
  - Monitoring
  - Control
  - Configuration

# Monitoring

- Watch instrumented health data and state changes
- Mechanism to listen for JMX notifications
- Autonomous reaction to adverse conditions
- Static event handlers (baked in code)
- Scripted event handlers
  - React to notifications using a scripting language
  - Allows agile adaptation to changes in business conditions and requirements
  - Extends system functionalities while core infrastructure remains intact

# Management & Spring AOP

- Increase separation of concerns
- Aspects can make infrastructural concerns, such as management, transparent to other concerns
- Aggregate instrumented data with aspects
- Push data to **MBeans** for instrumentation



# Spring AOP at Work

```
public class GreetingAspect {
  private GreetingMbean mbean;
  public void beforeGreeting() {...}
  public void afterGreeting() {...}
}
```

Management Bean

Aspect bean

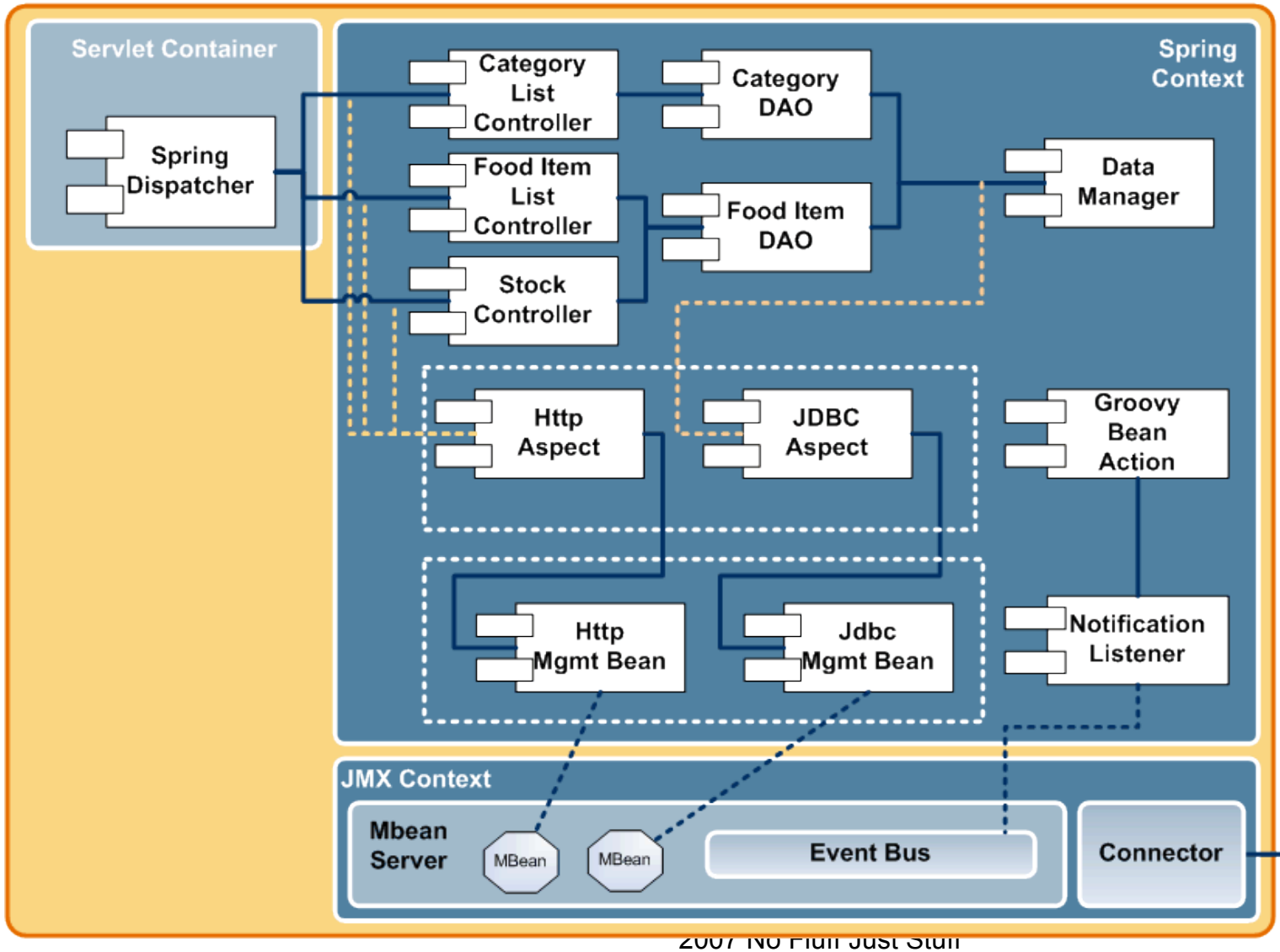
spring-context.xml

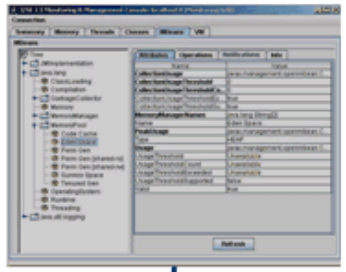
Intercepts

```
<bean id="aspectBean"
  class="demo.GreetAspect"/>
<aop:config>
  <aop:pointcut id="myPointCut"
     expression="execution(* demo.GreetServer.greet(..)) "/>
  <aop:aspect id="aspect" ref="aspectBean">
    <aop:before
      pointcut-ref="myPointCut"
      method="beforeGreeting"/>
    <aop:after
      pointcut-ref="myPointCut"
      method="afterGreeting"/>
  </aop:aspect>
</aop:config>
```

# Putting It All Together – Food Planet

- Components



# Summary

# Summary

- Log files are no longer enough for the critical applications
- JMX provides ability to monitor state changes in real time
- JMX technology includes a management server, event notification mechanism, monitoring service, and connectors for management clients
- It is available in JDK software starting with Java platform v.5
- JMX technology and JConsole management console provide a full-featured diagnostic, monitoring, and VM profiling tool
- Easily integrates into your coding efforts
- JMX is supported by nearly all major enterprise vendors

# For more Information

- JMX Technology
  http://java.sun.com/products/JavaManagement/
- Spring
  http://www.springframework.org/
- Groovy
  http://groovy.codehaus.org/
- Jconsole - http://java.sun.com/javase/6/
- MC4J - http://mc4j.org/
- Glassbox - http://glassbox.com/
- Broadway Project
  - http://code.google.com/p/broadway-monitor/

# Q/A

# Creating Manageable Systems With JMX

**Vladimir Vivien**
Simplius, LLC

vvivien@simpli.us
http://simpli.us/