# Scripting Your Way to Management with JMX & Groovy

**Vladimir Vivien**
Simplius, LLC

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

# Goal

Explore the use Groovy and the Java Management Extension API (JMX) to script runtime manageability into systems.

# Agenda

- Motivation
- Introduction to JMX Technology
- GroovyMBean
- JmxBuilder Management DSL
- Demo – Putting it all together
- 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

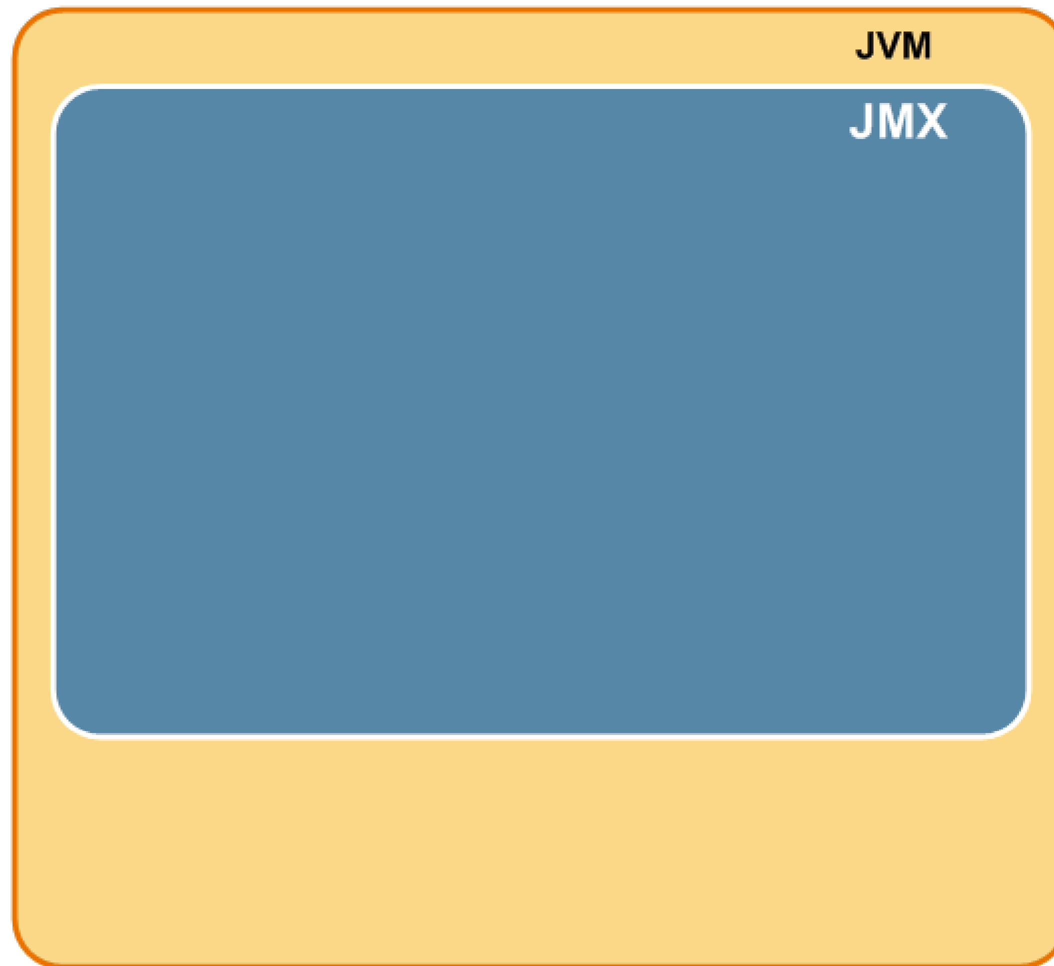
- Better visibility of system at runtime
- Go beyond simple event log files
- Ability to monitor
  - Expose states/health of application in real time
  - Take preventive measures where possible
  - React intuitively and quickly to changes & requirements
- Greater management
  - Interactively control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

# Introduction to JMX Technology

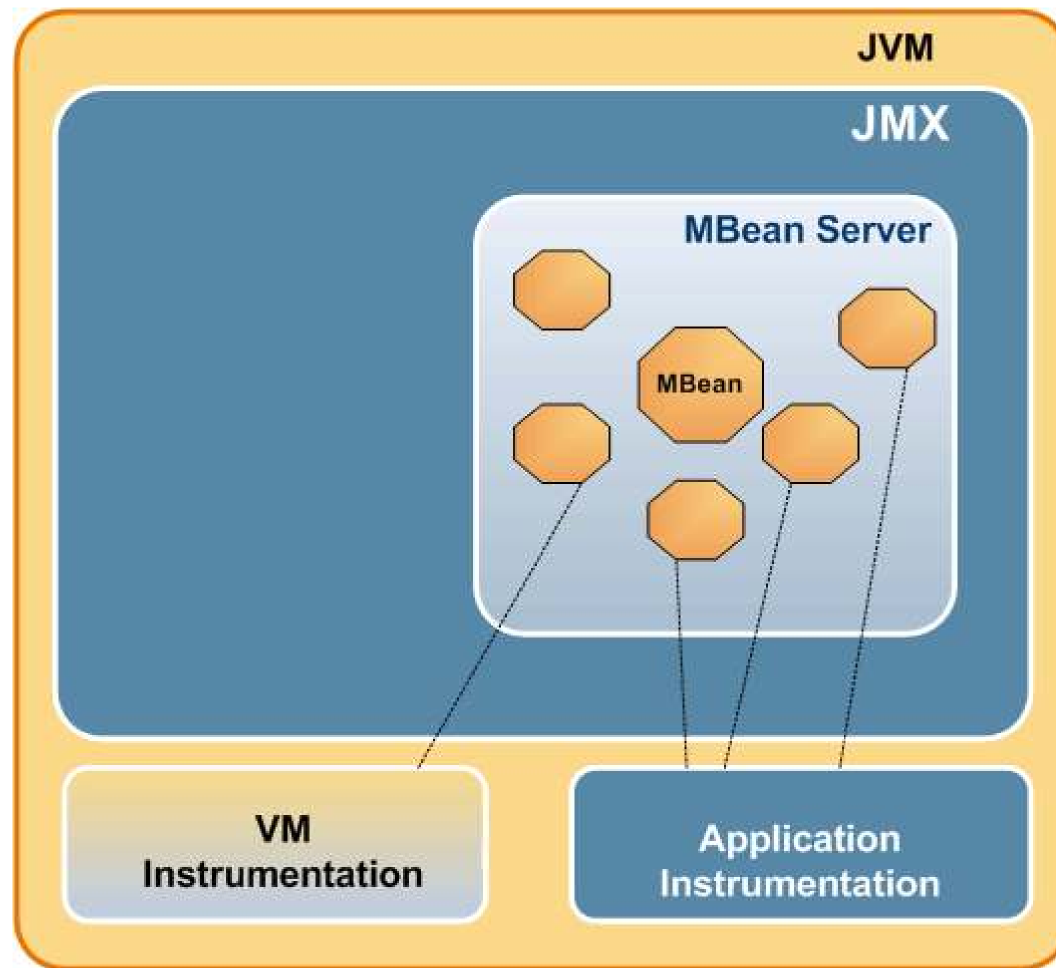
# What Is JMX?

- Java Management eXtension
- Standard API for runtime management & monitoring
- Mature API [originally (JSR)-3]
  - Uses a *client/server model*
  - *Server exposes* instrumented management *beans*
  - *Client uses connector* protocols to access instrumented data and management methods
  - Supports a *component-centric* programming approach with a full event model
- Part of core JDK starting with version 5
  - Exposes runtime VM profile information
  - Client tools include JConsole & Visual VM

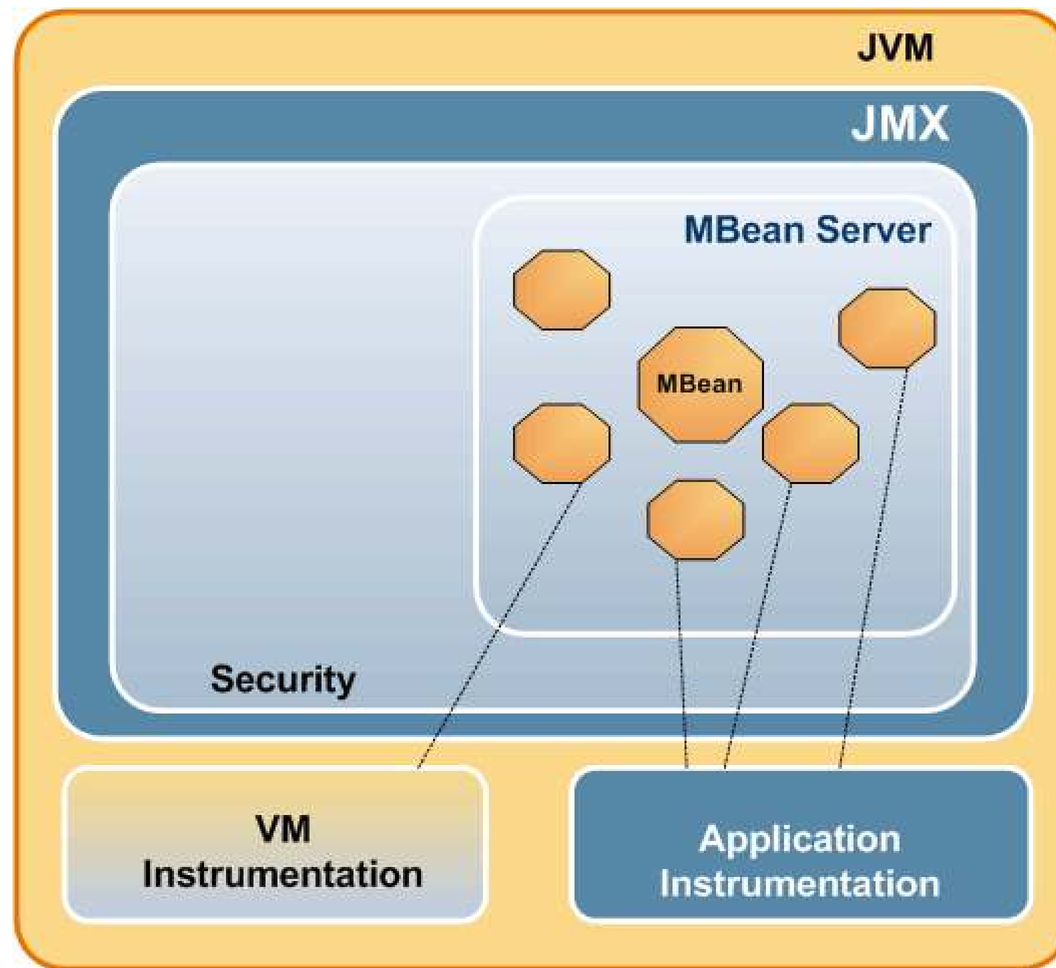
# JMX Architecture



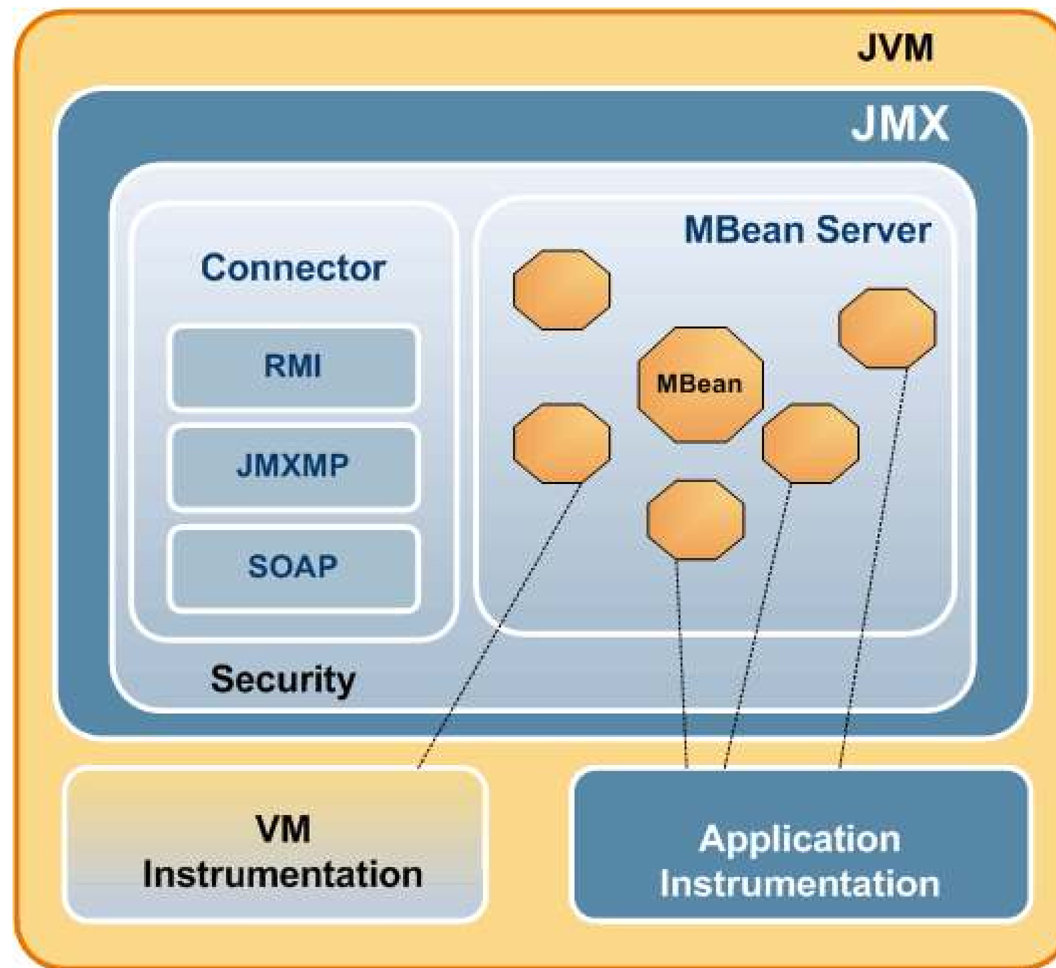
# JMX Architecture



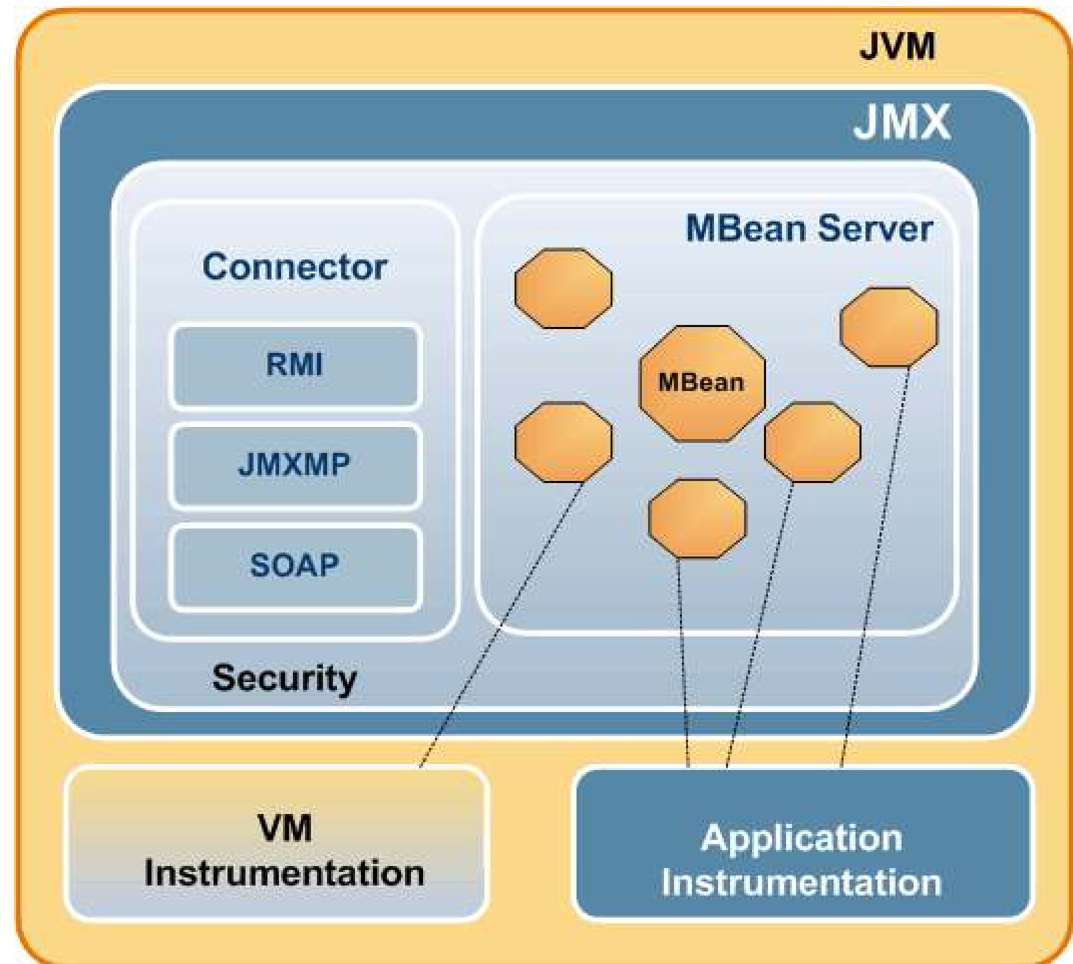
# JMX Architecture



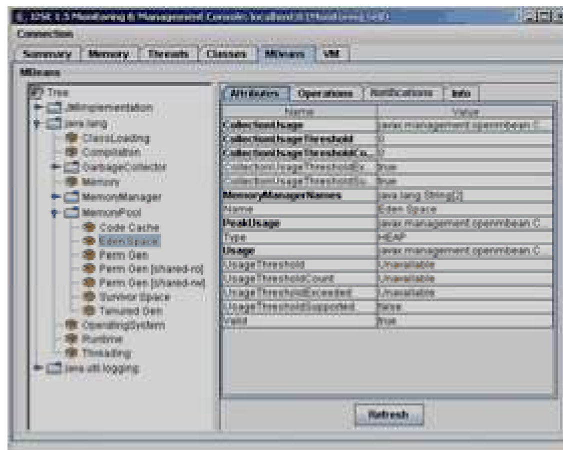
# JMX Architecture

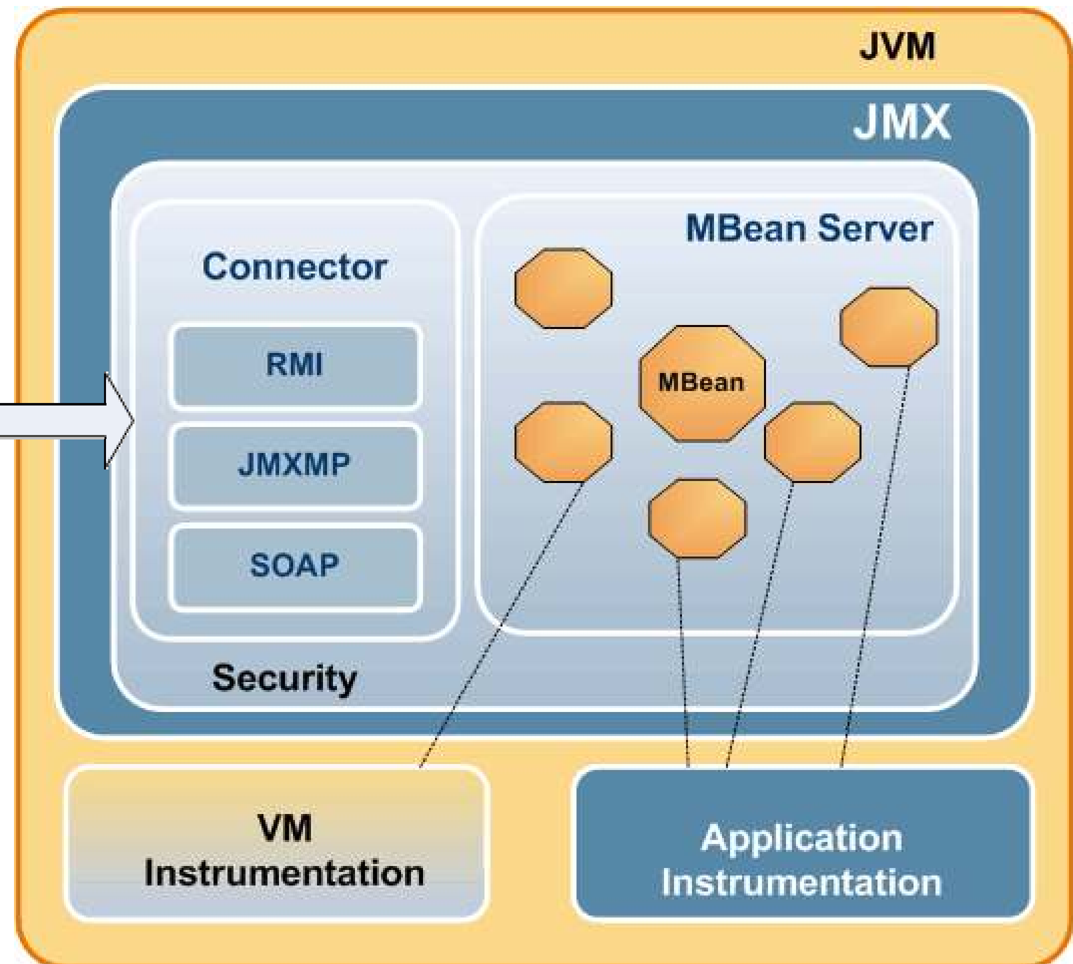


# JMX Architecture



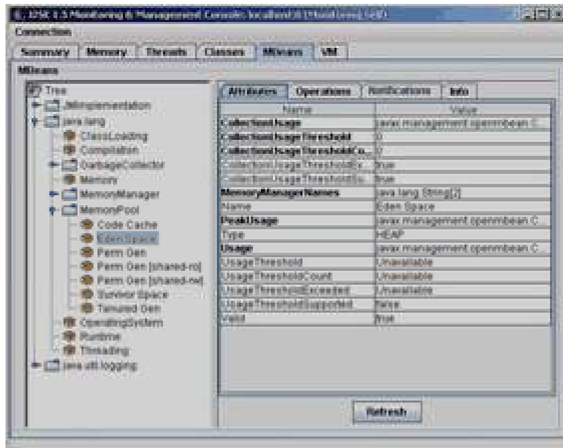
# JMX Architecture



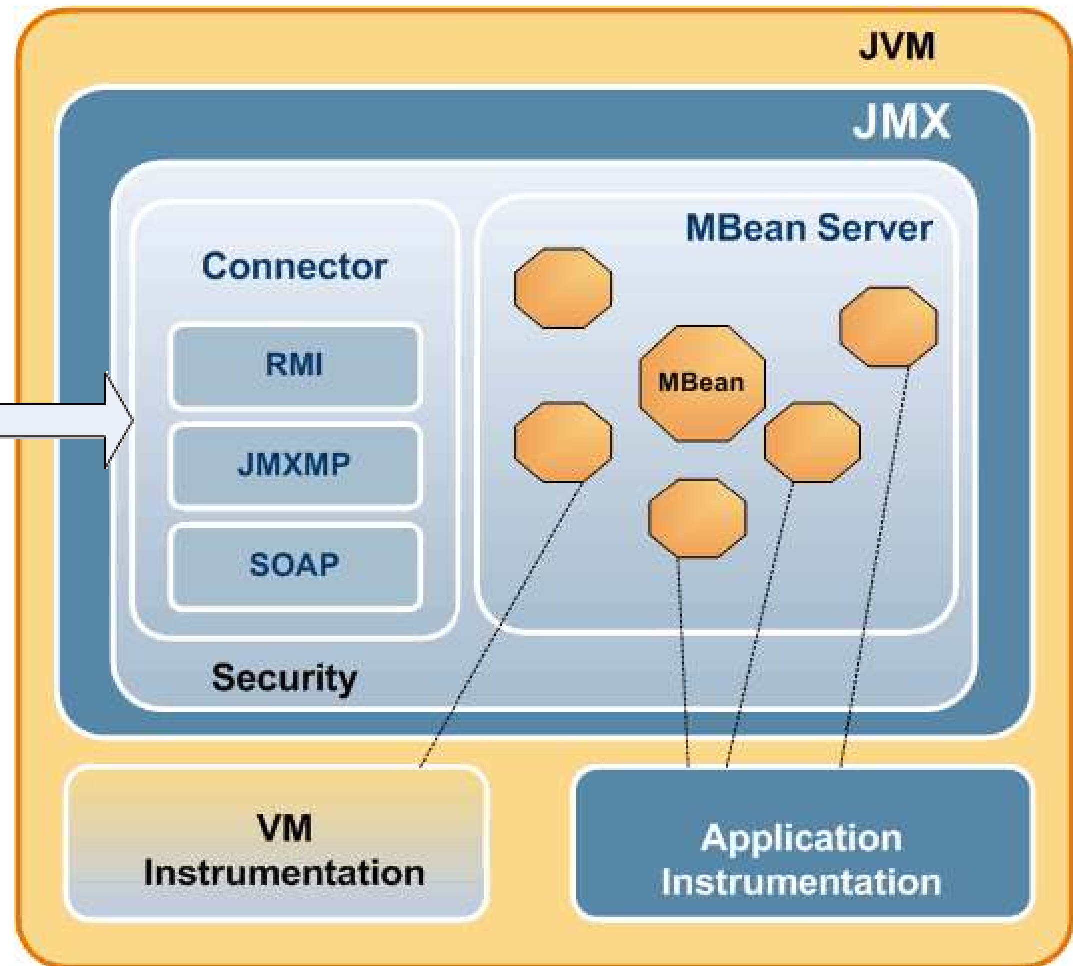


# JMX Architecture

**Management Console (Client)**



**Management Agent (Server)**



# Demo – Management Console

# The Management Bean (MBean)

- Simply put: a JavaBeans instance
- JMX MBean API
  - Registered with the MBean Server
  - Uses *ObjectName API*: a specially formatted string used as *identifier for the bean* in the server
    - **`[domain]:key=value[,key=value]*`**
    - *demo:type=Service,description=Object*
  - Exposes *attributes* (getters/setters)
  - Expose *operations* (interface methods)
  - *Send/consume messages* from JMX event bus

# Platform MBeans

- Starting with Java platform v.5, VM exposes numerous beans for platform instrumentation
- Platform beans include:
  - JVM class path information
  - Class loading information
  - Memory pool / consumption
  - Garbage collection
  - Operating system
  - Threads
  - And more...

# Useful JMX Tools

- Programmatic
  - Spring JMX
  - GroovyMbean class
  - JmxBuilder DSL
  - Commons Modeler
- Container/Declarative
  - JBoss Service Components
  - Geronimo Gbean
  - Glassfish Module
  - SpringSource Server
- Console Clients
  - JConsole
  - Visual VM
  - MC4J
  - Glassbox
  - JManage
  - JBoss Jmx-Console

# School of Management

- Management is an infrastructure concern
- Separate management from business layer (similar to DAO & logging)
- When/what to Manage
  - Faceless systems
  - Temporally durable processes & services
  - Controllable systems
- Types of Management
  - Monitoring
  - Control
  - Configuration

# Demo – JMX @ Work

# Groovy & JMX – GroovyMBean

# GroovyMBean

- Groovy class for client-side JMX
- Part of core Groovy
- Proxies JMX MBean attributes and operations
- Maintain natural Groovy syntax
- Hides JMX complexities, maintain bean idiom
- *No need for stubs on class path*
- Simplifies JMX API programmability
- Integrates well with other JMX helper API such as Spring JMX and JmxBuilder
- http://groovy.codehaus.org/Groovy+and+JMX

# GroovyMBean: What It's Good for

- Get programmatic handle on remote MBean
- Building management console
- Real-time gathering of resource metrics
- Runtime control of deployed resources
- Configuration updates sans system restart
- Shell scripts to control/monitor deployed resources (your sys admin will love you!)

# Using GroovyMBean

## Java – JMX API

```
{
  ObjectName name = new ObjectName(...);
  String attr = (String)server.getAttribute(name,'Symbol');
  server.setAttribute(name, new Attribute('Symbol','JAVA'));

  Object[] params = {new Integer(12)};
  String[] signature = {Integer.TYPE};
  server.invoke(name, 'sellStock', params, signature);
  // Not shown - tons of exception catching
...
}
```

## Groovy – GroovyMBean

```
{
  def tradeBean = new GroovyMBean(server,new ObjectName("..."))
  def attr = bean.Symbol
  bean.Symbol = 'JAVA'
  bean.sellStock(12)
}
```

# Demo – GroovyMBean @ Work

# Groovy & JMX
# JmxBuilder Management DSL

# JmxBuilder

- Created it over the last few months
- DSL for JMX API programming using Builder pattern
- Declaratively expose Java/Groovy objects as Mbeans
- No special interfaces, API or classpath restrictions
- Complete support for JXM descriptors
- Support class-embedded or explicit descriptors
- Seamlessly declare beans as event broadcasters
- Provide event listeners as inline closures
- Flexible registration policy for MBean
- http://code.google.com/p/groovy-jmx-builder/

# JmxBuilder: What It's Good for

- Create management agents (server-side JMX)
- Easily implement management infrastructure
- Generate GroovyMBeans from export of MBeans
- Access to JMX connectors and connector servers
- React to JMX events with POGO/POJO methods
- Send JMX event on POGO/POJO method invokation
- Intercept (and short-circuit) MBean operation calls
- Use with GroovyMBean for complete JMX programmability

# Using JmxBuilder (1)

## JMX Connector Server

```
// Returns a Connector Server with connection string
// "service:jmx:rmi:///jndi/rmi://localhost:1088/jmxrmi"

def server = new JmxBuilder().connectorServer(port:1088)
server.start()
```

## JMX Connector

```
// connects to an rmi-exported JMX server connector instance at
// "service:jmx:rmi:///jndi/rmi://localhost:1088/jmxrmi"
def client = new JmxBuilder().connectorClient(port:1088)
client.connect()
```

# Using JmxBuilder (2)

## Explicit Descriptor

```
// Given Java or Groovy class
public class TradingService {
  public void setSymbol(String lang){...}
  public void buyStock(int qty){...}
  Public void cancelTransaction(){...}
}
...
// Export POGO/POJO with JmxBuider, returns GroovyMBean[]
def bean = new JmxBuilder().export {
  bean(object:new TradingService(), name:"svc:type=trade..."){
    attributes {
      Symbol(desc:"Stock symbol to be traded")
    }
    operations{
      buyStock(desc:"Operation to sell stock"){
        quantity(desc: "stock quantity to buy",type:"int")
      }
      cancelTransaction(desc:"Cancels transaction in process.")
    }
  }
}
```

# Using JmxBuilder (3)

## Embedded Descriptor

```
// Groovy class definition only
public class TradingService {
  public void setSymbol(String lang){...}
  public void buyStock(int qty){...}
  Public void cancelTransaction(){...}

  static descriptor = {
    attributes {
      Symbol(desc:"Stock symbol to be traded", writable:false)
    }
    operations{
      buyStock(desc:"Operation to sell stock"){
        quantity(desc: "Stock quantity to buy",type:"int")
      }
      cancelTransaction(desc:"Cancels transaction in process.")
    }
  }
}
// Export POGO as Mbean, returns GroovyMBean[]
def bean = new JmxBuilder().export {
  bean(object:new TradingService(), name:"svc:type=trade...")
}
```

# Using JmxBuilder (4)

## Broadcast Event

```
...
def bean = new JmxBuilder().export {
  bean(object:new TradingService(), name:"svc:type=trade..."){
    ...
    notifications{
      onBuyStock(send: "stock.event.buy").call{event->
        ... // interceptor callback prior to trigger event
      }
    }
  }
}
```

## Listen for Event

```
def bean = new JmxBuilder().export {
  bean(object:new TradingService(), name:"svc:type=trade..."){
    notifications{
      listenFor(event: "stock.event.buy").call{event->
        log.info("Order to purchased placed ${event.source}”)
      }
    }
  }
}
```

# Demo – Putting it all together

# Summary

# Summary

- JMX lets you do runtime monitoring of your system
- JMX technology provides a API stack including server, event notification, connector APIs to build mgmt clients
- Groovy makes JMX programmability a breeze
- Use GroovyMBean to implement management consoles
- Use JmxBuilder to easily expose instrumented POJOs/POGOs as management components for your agent-side processes.
- JMX is supported by all major enterprise vendors.  Using Groovy you can create management scripts to configure, monitor, control, and manage enterprise resources.

# For more Information

- JMX Technology http://java.sun.com/products/JavaManagement/
- Groovy and JMX http://groovy.codehaus.org/Groovy+and+JMX
- Groovy JmxBuilder DSL http://code.google.com/p/groovy-jmx-builder/
- Spring and JMX http://static.springframework.org/spring/docs/2.5.x/reference/jmx.html
- Visual VM https://visualvm.dev.java.net/
- JConsole http://java.sun.com/javase/6/
- MC4J http://mc4j.org/
- Glassbox http://glassbox.com/
- JBoss JMX
  http://docs.jboss.org/jbossas/jboss4guide/r2/html/ch2.chapter.html
- Ruby JMX DSL
  http://developers.sun.com/learning/javaoneonline/2007/pdf/TS-9294.pdf

# Q/A

# Scripting Your Way to Management with JMX & Groovy

**Vladimir Vivien**
Simplius, LLC

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