

Monitoring and management using J2SE 5.0



Richard van der Laan

Agenda

- Monitoring and management
- J2SE 5.0 management features
- Java management extensions
- Demo: Remote manageable fridge
- *Earn a cold beer for each question*



Background (1/2)

- **Richard van der Laan**

- software engineer
- distributed, object-oriented development
- early adopter of Jini and OSGi technology
- last years involved in change of architectural styles towards service-orientation



Background (2/2)

luminis®

- Professional services company
 - area of distributed systems
 - object-orientation (most Java)
 - innovative projects
- Combined with
 - management and process improvement
 - training & coaching on all levels of software development



“Building the right system”
“Building the system right”
“Getting the system built”

Monitoring and management

- Access applications from remote locations
- Ability to remotely monitor, diagnose problems and perform system repairs
- Minimizing service and support by optimizing system self-awareness and remote capabilities



Available capability levels (1/2)



0. Access and remote collaboration

Remote connectivity and field service access

1. Collection and control

Remote tool operation, performance monitoring, data collection and storage



*“e-Diagnostics guidelines
for the semiconductor industry”*



Available capability levels (2/2)



2. Analysis

Automated reporting and advanced analysis

3. Prediction

Predictive maintenance, self diagnostics and automated notification



*“e-Diagnostics guidelines
for the semiconductor industry”*



Who can benefit

- Self-monitoring applications
- Application end users
- Administrators and management systems
- IT organization (*support engineers*)
- Development organization (*perhaps you*)



Important technology requirements

- Guaranteed *24x7* uptime
- Non-intrusive
- Lightweight
- Low datarate
- Safe and secure
- Useful in production environment



J2SE 5.0 management features

- JVM monitoring and management
- JConsole monitoring application
- Java Management extensions (JMX)



Important Java specification requests



- JSR-3: Java Management extensions
- JSR-160: JMX remote API
- JSR-163: Platform profiling architecture
- JSR-174: JVM monitoring and management

JVM monitoring and management (1/3)

- Monitor operating system resources
- View OS, runtime and compiler properties



OS properties

- os.name
- os.version
- os.arch

Compilation/JIT properties

- name of the compiler

JVM startup options

JVM system properties

- java.vm.vendor
- java.vm.version
- java.vm.name
- java.vm.specification.version
- java.vm.specification.vendor
- java.vm.specification.name
- java.class.path
- java.library.path

JVM monitoring and management (2/3)

- Monitor execution and synchronisation subsystems



Counters and accumulators

- total number of threads created and started
- number of live threads
- number of daemon threads

Thread general info

- thread identifier
- thread state:
 - created
 - running
 - blocked
 - terminated
 - ...

JVM uptime

- total time since started



JVM monitoring and management (3/3)

- Monitor classloading and memory subsystems



Counters and accumulators

- current number of loaded classes
- total number of loaded classes since started
- total number of unloaded classes since started

Heap info

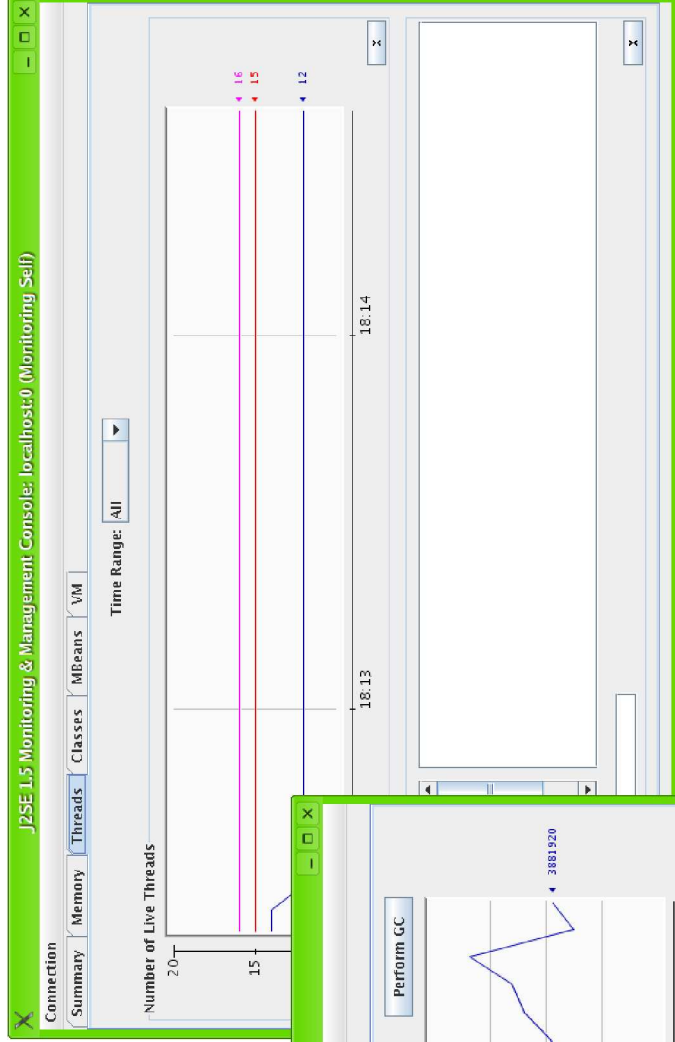
- initial amount of allocated memory
- amount of used memory
- maximum amount of memory

Garbage collector

- current number of collections
- accumulated time in GC

JConsole

“View memory usage”



*“View number
of threads”*

Java management extensions

- Enables application management and monitoring
- Specification defines:
 - Layered architecture
 - Design patterns
 - Interfaces
 - Services

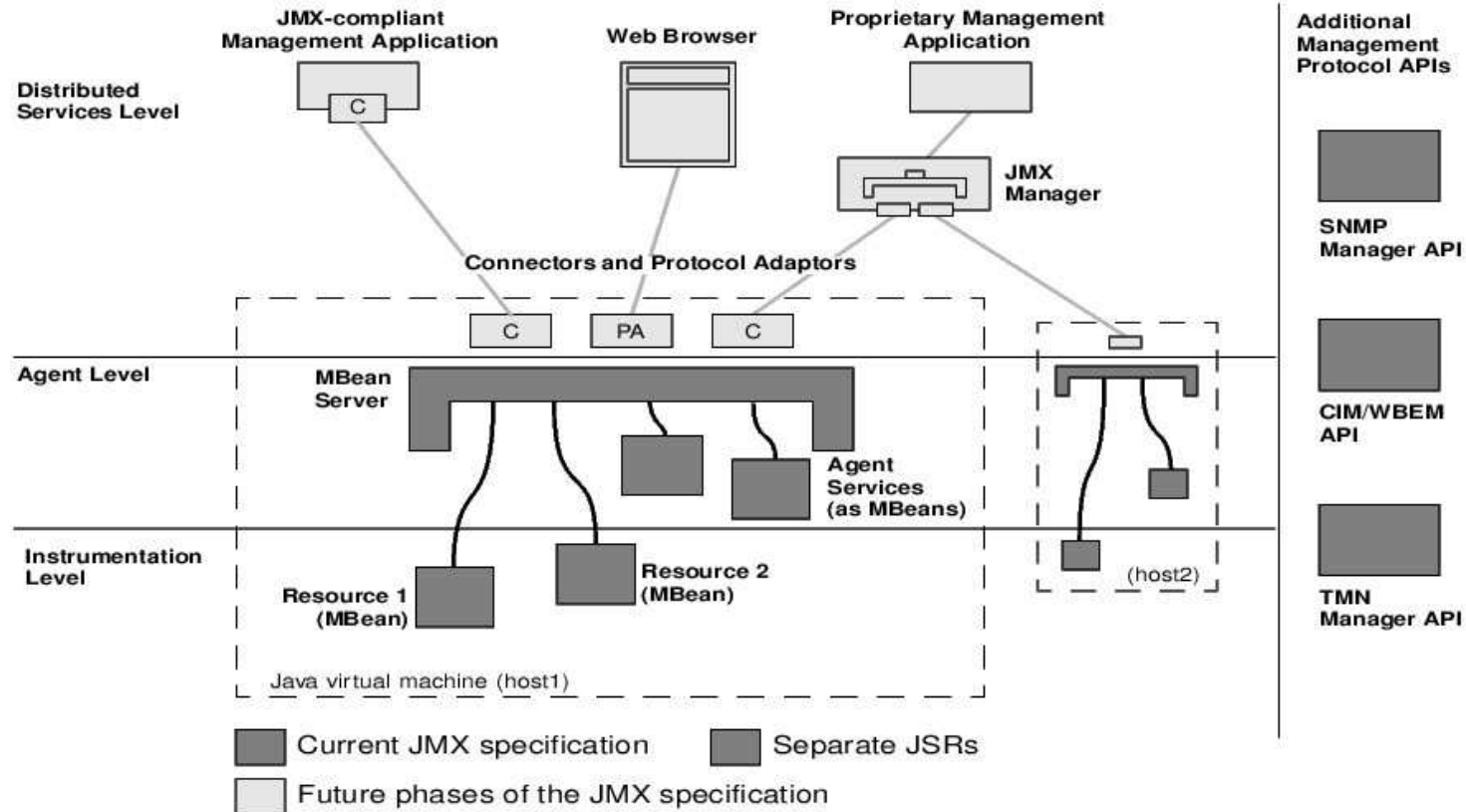


Architectural layers

- Instrumentation level
- Agent level
- Distributed services level



Architectural overview



Instrumentation level (1/7)

- Provides the instrumentation to create manageable resources
- Resources are wrapped in Managed Beans
- Use of design patterns derived from the JavaBeans component model



Instrumentation level (2/7)

- MBeans expose attributes and operations

```
public class MyResourceBean {  
    ..  
    public String getName() {..}  
    public void setName(String name) {..}  
    public boolean isStarted() {..}  
    public void start() {..}  
    public void stop() {..}  
}
```

Instrumentation level (3/7)

- MBeans can emit and receive notifications

```
public class MyResourceBean implements NotificationEmitter {  
    public MBeanNotificationInfo[] getNotificationInfo() {...}  
  
    public void addNotificationListener(NotificationListener l,  
        NotificationFilter f, Object handBack) {...}  
  
    public void removeNotificationListener(NotificationListener l) {...}  
    public void removeNotificationListener(NotificationListener l,  
        NotificationFilter f, Object handBack) {...}  
}
```

Instrumentation level (4/7)

- MBeans can provide metadata describing the management interface

```
public class MBeanInfo {  
    public String getClassName() {...}  
    public MBeanNotificationInfo[] getNotifications() {...}  
    public MBeanAttributeInfo[] getAttributes() {...}  
    public MBeanConstructorInfo[] getConstructors() {...}  
    public MBeanOperationInfo[] getOperations() {...}  
    public String getDescription() {...}  
}
```

Instrumentation level (5/7)

- Different MBean types to meet different needs for instrumentation:
 - Standard MBeans
 - Dynamic MBeans
 - Open MBeans
 - Model MBeans



Instrumentation level (6/7)

- Flexible interface evolvability using Dynamic MBeans

```
public class MyDynamicResource implements DynamicMBean {
    public MBeanInfo getMBeanInfo() {...}
    public Object getAttribute(String name) {...}
    public AttributeList getAttributes(String[] names) {...}
    public void setAttribute(Attribute attribute) {...}
    public AttributeList setAttributes(AttributeList attributes) {...}
    public Object invoke(String action, Object[] params,
        String[] signature) {...}
}
```

Instrumentation level (7/7)

- Open MBeans
 - Dynamic MBeans which expose only *basic data types* for easy management integration
- Model MBeans
 - Dynamic MBeans supplied by JMX agents offering *default and configurable behavior*



Agent level (1/4)

- MBean server
 - Secure permission based access
 - Instantiate and register MBeans
 - Lookup MBeans
(based on object name or attribute value)
 - Access MBeans
(through their management interface)



Agent level (2/4)

- Connectors
 - Access agents from remote management applications (*like RMI and JMXMP connectors*)
- Protocol adapters
 - Provide a view through specific protocols (*like SNMP adapter*)

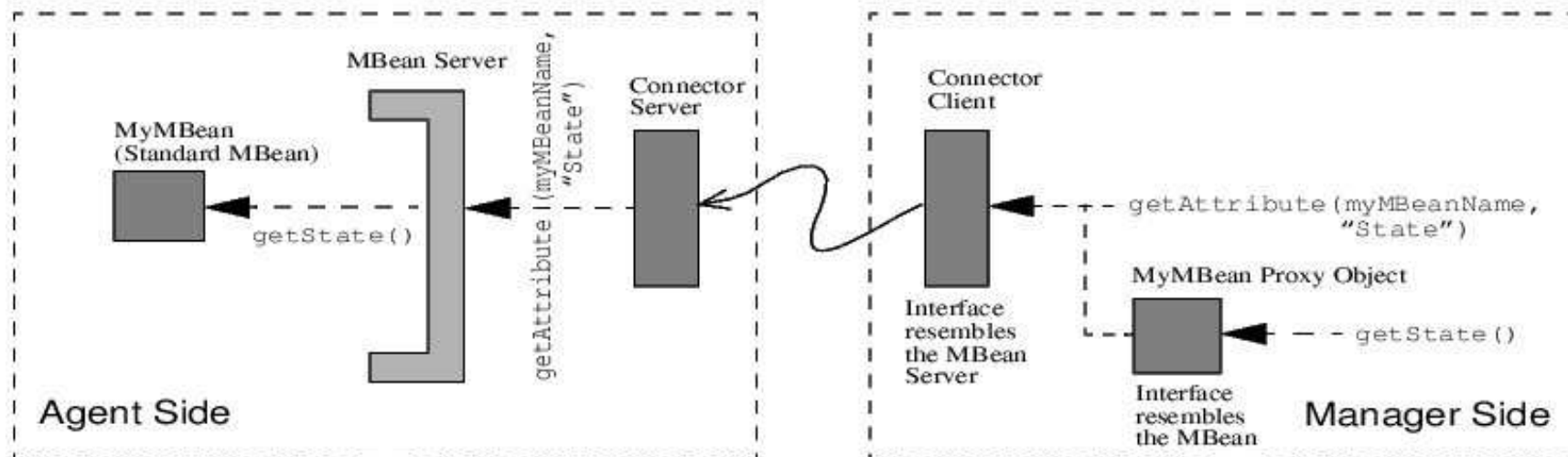


Agent level (3/4)

- Connecting agent and manager



```
service:jmx:jmxmp://myhost:8765
```



Agent level (4/4)

- Other available agent services
 - M-lets (*dynamic classloading*)
 - Monitors (*MBean attribute observers*)
 - Timers (*time-based scheduling mechanisms*)
 - Relation service
(*define associations between MBeans*)



Benefits of the JMX architecture

- Use without heavy investment
- Scalable architecture
- Easy integration of existing standards
- Embraces existing Java technologies
- Defines only the necessary interfaces



JMX in practice (1/2)

- J2SE 5.0 standard feature
- JBossMX
- MX4j
- X-mojo
- EJTools



JMX in practice (2/2)

- Java Dynamic Management Kit
- AdventNet SNMP adaptor
- TMX4J Tivoli Monitoring
- CA Unicenter



Remote manageable fridge (1/2)

- Case study for bridging transport and logistic markets
- Monitor freight temperature and trailer door status
- Monitor truck speed, torque and fuel consumption



Remote manageable fridge (2/2)

- Equipped with industry standard CAN-bus sensors
- J2SE 5.0 Linux platform
- JMX remote monitoring
- Full OSGi compliant gateway featuring remote component life-cycle management





info@luminis.nl
www.luminis.nl