Managing JMX Views with JMXPrism

Nicolas Rivierre, Thierry Coupaye - France Telecom R&D
5th Fractal Workshop – 3/7/06, Nantes

Objective

Simple JMX mediation layer
Between managers (console, programme) and managed systems
• To adapt legacy JMX management profiles (JVM, J2EE, ad-hoc…)
• To instrument Fractal applications (FractalJMX)
Outline

Problem statement
- Background on JMX
- Additional requirements

Proposed solution
- Approach
- Design

Examples
- AutoTune, GIS

Conclusion

Background on JMX

Framework and API
to build manageable Java applications
and integrate them with existing management solutions

Architecture (core)
- Instrumentation
  - MBeans (resources)
- Agent
  - MBeanServer (registry, ObjectName)
    - To let managers invoke op-att, rcv/snd notif
  - Agent services
- Distribution
  - Remote managers
  - Adaptors/connectors (http, snmp, rmi, WS...)
Background on JMX (2)

Agent services
- Timer, relation, Monitors...

Management profiles
- Ex. Monitoring and Management Specification for the JVM (JSR 174)
  - Threads (counters, state, locks..), Mem (heap, pools...), Class loading, GC...
  - Ex. Thread stat. (Jconsole)

On-demand, low overhead, JMX/SNMP access...

Additional requirements

JMX
- De-facto management standard, relatively complete on the server side...

However, on the manager side
- How to avoid ad-hoc design?
  - Multiple connexions
  - Additional services (not planned)
  - Feedback loop
  - Transformation (filtering, renaming, aggregation...)

- How can dynamic and non-intrusive properties be obtained?
  - To adapt legacy JMX management profiles (JVM, J2EE, ad-hoc...)
  - To instrument Fractal applications
  - Etc.
Outline

Problem statement
  - Background on JMX
  - Additional requirements

Proposed solution
  - Approach
  - Design

Examples
  - AutoTune, GIS

Conclusion

Approach

Simple mediation layer (JMXPrism)
  - Between managers (console, programme) and managed systems

Basic features
  - Unique access point (JMXPrism MBeanServer)
  - Configure/deploy custom JMX management profiles
    - Assembly of components (views, sources, connectors, bus…)
    - To populate JMXPrism MBeanServer with relevant Mbeans
  - Dynamic and non-intrusive
  - Bus to handle MBeans notifications
Design

➢ Instrumentation layer → pure JMX
  
  MBeans (static/dynamic...) represent resources, services...
  
  in a uniform way (attributes, operations, notifications)

➢ Information model

➢ ”Interaction model”

➢ Framework
  
  Fractal components

  Fractal instrumentation

Design – information model

Management views to

Describe relevant information → e.g. on legacy rmi/mejb JMX profiles

Cnx + filters (MBeans, operations...) + additional services (monitors...)

Deploy this information → i.e. to expose MBeans in JMXPrism MBeanServer

Proxy MBeans (cascading MBeanServers)...

Definition <itf>

➢ View: expose(mbs)/unexpose(mbs)

➢ Composite (hierarchical name space, forwards expose)

➢ [Attributes] (e.g. sources, filters, monitors...)
**Design – "interaction model"**

Managers use paths in the hierarchical name space of views

*Admin*: `expose(path) / unexpose(path)`

Avoid complex navigation & API dependencies (MBeanServer...)

---

**Design – Framework**

*Fractal-Based*

No Fractal implementation dependencies

*Component library (to extend)*

- **Agent** (`MBeanServer`, *Admin*)
- **Connectors** (server side)
- **Management views**
- **Sources** (client side)
- **Bus** (MBeans notifications)
- **Listener/effectors**

*ADL, API*

**or JMX (self-instrumentation)**
Notes on Fractal instrumentation

Additional functionality of JMXPrism (formerly FractalJMX)

Dynamic instrumentation of Fractal applications

Principe

1 – added to any (Java) Fractal application
   Subcomponents: Agent, Connectors (rmi, http, ...)

2 – expose Introspect the component structure
   Implicit view: super-component(s) & sub-components (recursively)

3 - Create & register (dynamic) MBeans
   Fractal server interfaces (fct, att, ctrl)

4 - Attributes
   JMX monitors & Filters to avoid flooding

E.g. Comanche

<?xml version="1.0" encoding="UTF-8"?>
<definition name="comanche.Comanche" extends="comanche.ComancheType">
  component name="fe" definition="comanche.Frontend"/>
</definition>

Outline

Problem statement
  Background on JMX
  Additional requirements

Proposed solution
  Approach
  Design

Examples
  AutoTune, GIS

Conclusion
AutoTune

Simple control of available resources (JVM & app. Monitoring)

Grid Information System (GIS)

Monitoring Service for large-scale grids
- Clusters of clusters of heterogeneous servers
- Software probes deployed on every server
  monitoring performance metrics (CPU load, memory usage, number of jobs...)
- Human & automatic resource management

JMXPrism usage: JMX cluster management profile

France Telecom
Research & Development
Outline

Problem statement
  Background on JMX
  Additional requirements

Proposed solution
  Approach
  Design

Examples
  AutoTune, GIS

Conclusion

Conclusion

JMXPrism: a simple JMX mediation layer
  The framework is component-based, (relatively) easy to use & extend
  It is suited for management profile adaptation (JMX oriented)
    Not to model the managed systems as a component-based architecture
  Its prototype library & functions are simple but have been tested in real cases

Status
  2003: FractalJMX (ObjectWeb)
    Non-intrusive, dynamic instrumentation of (Java) Fractal applications
    No impl. dependencies except optional Julia controllers (basic exponential smoothing)
  2004: JMXPrism v0
    Extensions to adapt legacy management profiles (JVM, J2EE, ad-hoc…)
  2005: use cases (GIS, Speedo)

Future works?
  Re-factoring, extensions, synchronisation, aggregation….
Questions?