Appendix A

Java management extensions (JMX)

The query optimization technique proposed in this thesis project suggests optimal solutions taking in to account the execution environment, specifically, an estimation about the computational resources available for the query plan execution. Also it is required to keep track about the computational resources consumed during the plan execution. Resources can be physical, for example small network devices, or logical, for example installed applications.

SQuO involves a simple implementation for resource measurement. Currently these resources involve i.e. memory and CPU consumption, time, among others. However, the extension and application of this solution to different areas is a perspective (i.e. coordination of services). For monitoring and management of most complex resources, the Java Management Extensions (JMX) is a feasible option since it offers a framework for it.

Overview

The Java Management Extensions (JMX) framework is a standard developed through the Java Community Process for managing and monitoring applications and services. It defines a management architecture, design patterns, APIs, and services for building web-based, distributed, dynamic, and modular solutions to manage Java-enabled resources [OC10].

This framework provides Java developers with the means to instrument Java code, create smart Java agents, implement distributed management middleware and managers, and easily integrate these solutions into existing management and monitoring systems. The agents know how to get the state of the resources, and are permitted to change them if exposed in a writable fashion. The agents are then accessible through a distributed layer (i.e. the Simple Network Management Protocol - SNMP). This allows to control the settings of the devices without having to run to equipment closets or create custom management tools for applications [OC10][Ma04].

The JMX APIs make it possible to add manageability to Java-enabled equipment, from web phones to set-top boxes to network devices and servers. Using JMX technology to manage applications and services increases their value to vendors and clients by making applications easier to install, configure, and maintain.

There are two JAR to use this tool, lib/jmxri.jar and lib/jmxtools.jar. The jmxri.jar file represents the reference implementation. These are the standard JMX classes, and can be found in the javax.management package. The jmxtools.jar file is the JMX toolkit. These represent unsupported classes that Sun provides for development.

Advantages

The JMX technology enables Java developers to encapsulate resources as Java objects and expose them as management resources in a distributed environment. The JMX specification lists the following benefits to using it to build a management infrastructure [OC10] [Ma04]:

- Manages Java applications and services without heavy investment. JMX architecture relies on a core managed object server that acts as a management agent and can run on most Java-enabled devices. Java applications can be managed with little impact on their design.

- Provides scalable management architecture. A JMX agent service is independent and can be plugged into the management agent. The component-based approach

enables JMX solutions to scale from small footprint devices to large telecommunications switches.

- Can leverage future management concepts. It can implement flexible and dynamic management solutions..

- Focuses on management. While JMX technology provides a number of services designed to fit into a distributed environment, its APIs are focused on providing functionality for managing networks, systems, applications, and services.

Architecture

JMX technology provides a tiered architecture where managed resources and management applications can be integrated in the plug-and-play approach as shown in Figure A.1. There are three levels in this architecture: instrumentation, agent, and manager [OC10] [Ma04].

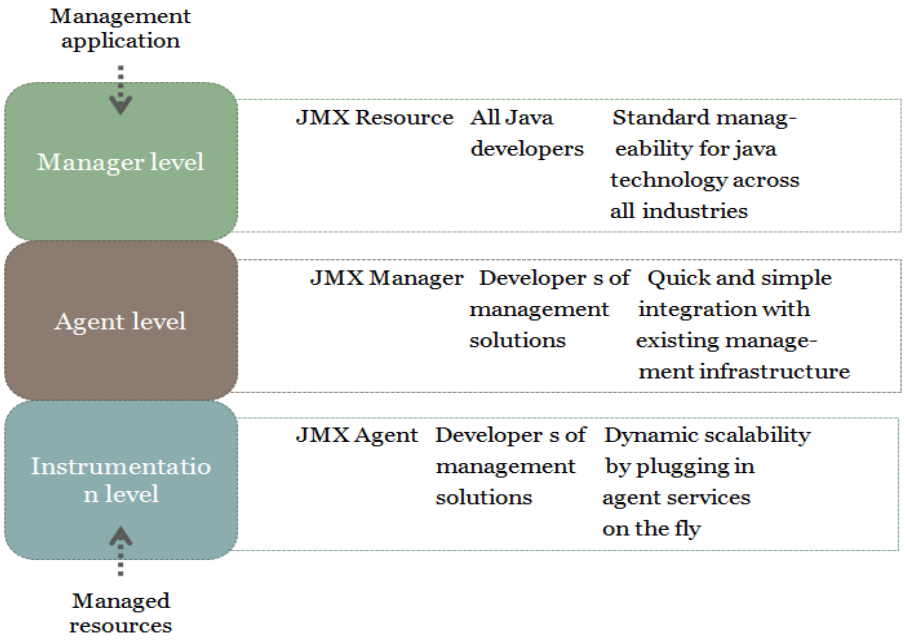

Figure A.1 JMX Architecture

Instrumentation Level

This level provides a specification for implementing JMX technology-manageable resources, which can be an application, service, device, or user. A resource is manageable if it is developed in Java (or provides a Java wrapper) and has been instrumented so that it can be managed by JMX-compliant applications. The key Components at the instrumentation level are the MBeans, the notification model, and the MBean metadata classes.

- MBean object. An MBean is a Java object that implements a specific interface. The management interface of an MBean is represented as: (1) valued attributes that can be accessed; (2) operations that can be invoked; (3) notifications that can be emitted; and (4) the constructors. There are four types of MBeans: standard, dynamic, open, and model. Standard MBeans are Java objects that conform to certain design patterns (for example, they must have a constructor and setter/getter methods). A dynamic MBean conforms to a specific interface that offers more flexibility at runtime. An open MBean is a dynamic MBeans that rely on basic data types for universal manageability; they are self-describing for user-friendliness. Finally a model MBean is a dynamic MBeans that are fully configurable and self described at runtime.

- Notification Model. JMX technology defines a generic notification model based on the Java event model. It lets developers build proactive management solutions. Using notifications, JMX agents and MBeans can send critical information to interested parties such as management applications or other MBeans.

- MBean Metadata Classes. These classes contain the structures to describe all components of an MBean's management interface: its attributes, operations, notification, and constructors. For each of these, the metadata include a name, a description and its particular characteristics (for example, an attribute is readable, writeable, or both; for an operation, the signature of its parameter and return types).

Agent Level

This tier contains the JMX agents used to expose the MBeans. It provides a specification for implementing agents, which control the resources and make them available to remote management applications. Agents are usually located on the same machine as the resources they manage, but this is not a requirement. The JMX agent consists of an MBean server and a set of services for handling MBeans. Managers access an agent's MBeans and use the provided services through a protocol adaptor or connector. But note that JMX agents do not require knowledge of the remote management applications that use them. The main components at the agent level are the MBean Server and Agent Services.

- MBean Server: A registry of objects that are exposed to management operations in an agent. Any object registered with the MBean server becomes visible to management applications. However, note that the MBean server only exposes an MBean's management interface and never its direct object reference. Any resources that you want to manage from outside the agent's JVM must be registered as an MBean in the server. The server also provides a standardized interface for accessing MBeans within the same JVM, giving local objects all the benefits of manipulating manageable resources.

- Agent Services: Objects that can perform management operations on the MBeans registered in the MBean server. By including management intelligence in the agent, JMX helps you build more powerful management solutions.

Manager (or Distributed Services) Level

This tier contains the components that enable management applications to communicate with JMX agents. It provides the interfaces for implementing JMX managers, and defines the management interfaces and components that operate on agents. Such components provide an interface for a management application to interact with an agent and its JMX manageable resources through a connector, and also expose a management view of a JMX

agent and its MBeans by mapping their semantic meaning into the constructs of a data-rich protocol (such as HTML) [OC10].

JMX comprises a separate package for each tier of the management architecture. The instrumentation tier will be free, and other tiers can be built from public specifications or reference implementations available under Sun Community Source License. Alternatively, you can purchase commercially supported products.