Apache Geronimo With A Monitor Component (JMX and JConsole)

The purpose of this article is to show the possibilities of server-side monitoring using Geronimo, a Java™ 2 Platform Enterprise Edition (J2EE) application server. Monitoring an application server from inside saves network traffic, since monitored information can be analyzed, filtered, summarized, and set into an application specific context inside the server. For example an application server could send an e-mail when the response time of our online shop gets unacceptable big. A simple web application has been used to develop a server-side monitoring component that monitors three servlets and gives alarm when the overall average processing time exceeds a certain given threshold.

What is needed

- JDK 5.0 (because of the used JConsole tool; it is not included in JDK1.4): http://java.sun.com/javase/downloads/index.jsp
- Standard Geronimo: http://geronimo.apache.org/downloads.html
- Sample Code: http://www.informatik.hs-furtwangen.de/~reich/Geronimo/GMontiorSample.zip (Contents: Sources, shell scripts, archives, README, etc.)

Introduction

A lot of articles can be found in managing application servers with remote clients using the Java Management Extensions (JMX; http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/). These management client applications describe how to connect management clients to a server and how to retrieve information from the server applications. While much attention has been given to the client-side aspects of JMX, very little consideration has been given to the server-side challenges of developing and deploying management beans (MBeans). The reason lays in the difficulties of development and integration of such components. Often it is not possible at all. With the appearance of Geronimo the server-side monitoring by implementing MBeans for monitoring servlets or EJBs is simplified.

This article shows how to monitor the application server Geronimo from inside and how detailed information querying the MBeans can be analyzed, grouped and generated to meta data inside the server. This saves bandwidth between the management client and the server and allows to build a more efficient controlled application server with the monitor component inside.

To keep it simple the Geronimo monitoring component is investigating the processing time of three servlets. The average of all three processing times is built and an alarm is generated if the overall processing time is greater then a pre-defined value. How the developed monitoring component is integrated into the Geronimo architecture and how it is deployed and managed by the JConsole http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/ is content of this paper.

Overview about MBeans and GBeans

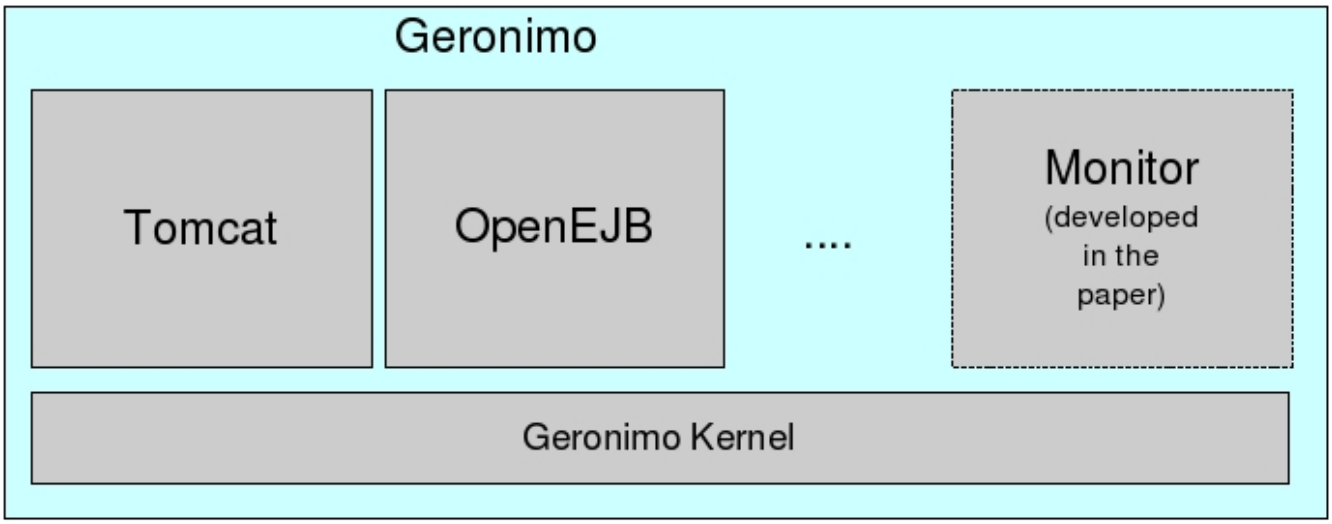
This section gives a short introduction of the management standard JMX (http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/) with its fundamental beans, the MBeans. It explains that Geronimo can be extended by new components, if they are GBeans.

Java Management Extension

Java Management eXtension (JMX) (http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/) standardizes the managing and monitoring of applications and services. It enables developers to write management programs for their applications in a vendor/neutral fashion. Another important standard in the area of J2EE is JSR-77 (http://jcp.org/en/jsr/detail?id=77). JSR-77 is a standard model for managing the J2EE platform and allows application server vendors to present performance metrics in a standard way. It defines a set of standard metric types that can be used to monitor J2EE platforms. The following types of metrics are defined in JSR-77: range statistics, boundary statistics, bounded range statistics, count statistics, and time statistics. Fundamental to JMX is the management bean, MBean. There are four types of MBeans (Standard, Dynamic, Model, Open), and each provide a different level of sophistication for management and monitoring (see http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/ for more details). In this paper, MBeans are used for the monitoring of servlets and for controlling the GBean monitor component.

Geronimo hosting our Monitor Component

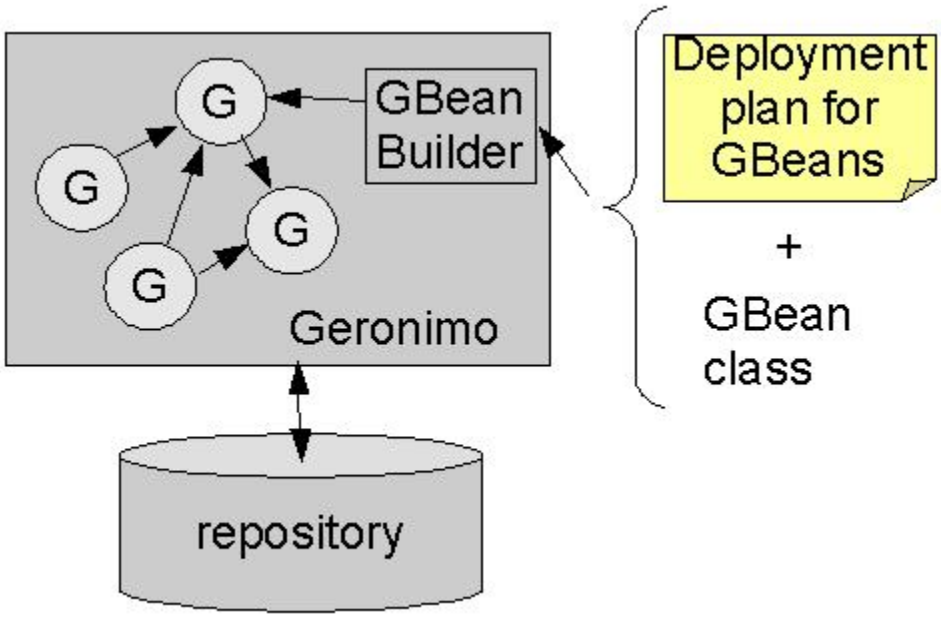
Geronimo is in the first place a Java™ 2 Platform Enterprise Edition (J2EE) application server, but can be seen as a general service container. The main focus of developing Geronimo was the managing and scaling of application servers. Geronimo's intention was never to re-implement a servlet or an EJB container, but to use existing open source applications (mainly from Apache http://www.apache.org/) whenever possible, plug it together and build a new application server.



The standard Geronimo distribution comes with Apache Tomcat http://tomcat.apache.org/ and EJB container (OpenEJB http://incubator.apache.org/openejb/) component. Extending Geronimo by a monitoring component, developed for this paper, is like Tomcat, or any other component as long as they are GBeans, as shown in the Figure: #Geronimo extended by a monitor component. The fundamental entity within Geronimo are Geronimo Beans (GBeans).

Geronimo Beans

Everything in Geronimo is basically a Geronimo Bean (GBean). Geronimo's kernel handles these GBeans and stores them in it s repository. By default Geronimo uses not a database but a file directory named `repository`.



Users can install their own GBeans by describing them with a deployment plan and deploy them into Geronimo by using the GBeanBuilder (see Fig. #Geronimo GBeans). Every GBean must implement the `GBeanLifecycle` interface. This interface defines three methods: `doStart()`, `doStop()`, `doFail()` as you see in the class diagram (Fig. #Class diagram of the management component example). The `GBeanLifecycle` interface is the contract between the Geronimo plug-in framework and our application, the GBean. The Geronimo framework uses dependency injection (see Fowler: Inversion of control containers and the dependency injection pattern, Frod: Dependency injection in apache geronimo, part 1, Dependency injection in apache geronimo, part 2) to set GBean parameters during the deployment process. There are two ways to inject information during the deployment process of GBeans:

1. getter/setter injection: framework injects information using the setter and getter methods of the GBean.
2. constructor injection: framework injects information using the constructor of the GBean, when it is instantiated.

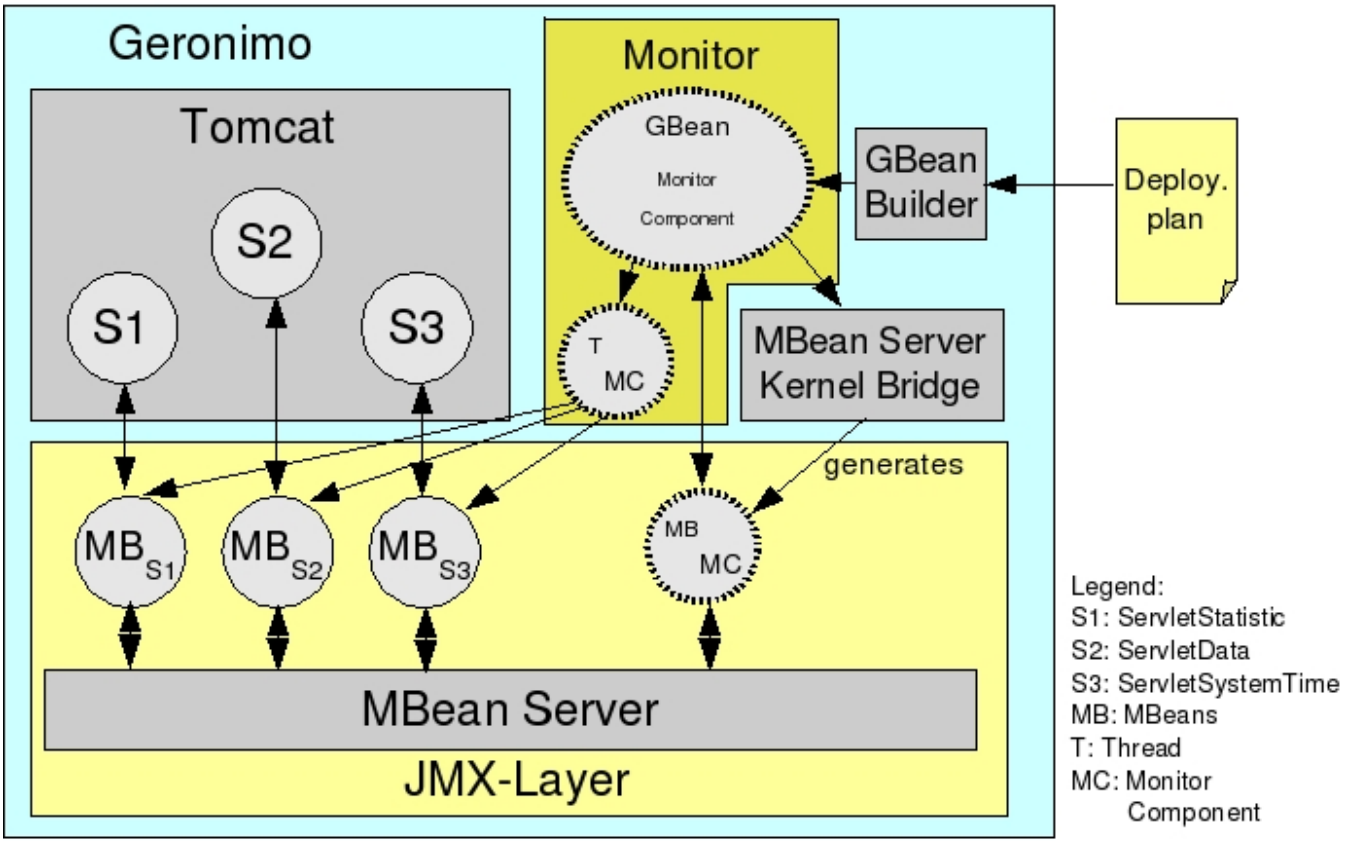
The example of this article is using constructor injection to pass the servlet MBean names and the MBean server reference to the monitor GBean.

The Server-Side Monitor Component

Subject of the section is a detailed architecture description of how the GBeans and MBeans are interacting. Discussed are the developed monitor classes and how to deploy the monitor component into Geronimo. Finally the monitor component is tested using the JConsole.

Geronimo Architecture Overview

First an overview about the integration of the monitor component into Geronimo's architecture is given in Figure #Geronimo architecture with integrated monitor component. It shows the JMX-Layer with the MBeanServer and the MBeans, the Tomcat component and the monitor component.



Geronimo registers each GBean as a MBean with the MBean Server, when the server is started. Tomcat http://tomcat.apache.org/, for example, consists of several GBeans and therefore consists of several MBeans (TomcatWebContainer, TomcatWebConnector, TomcatEngine, TomcatJAASRealm, etc.). For each servlet (S1, S2, S3) in Fig. #Geronimo architecture with integrated monitor component there exists a MBean (MB_{S1}, MB_{S2}, MB_{S3}) with information defined in the JSR-77 specification. The dashed line objects are instantiated, when the sample monitor component (Servlet Monitor GBean) is deployed into Geronimo using the "GBean Builder". The MBeanServerKernelBridge registers each loaded GBean as a MBean (MB_{MC}) at the MBean server. The instantiated thread (T_C) retrieves information from the servlet MBeans (MB_{S1}, MB_{S2}, MB_{S3}).

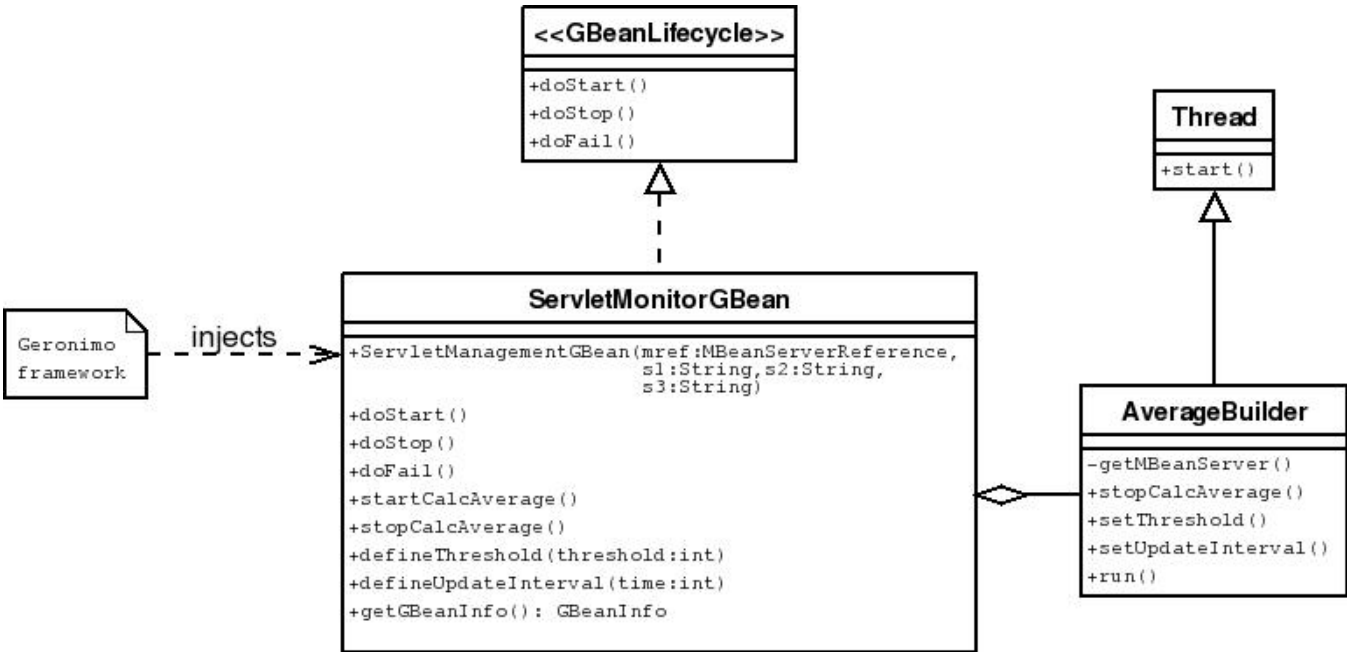
Servlets under observation

It is not the content of this article to show how to develop and deploy servlets. There is a simple security servlet example in Geronimo's user guide http://cwiki.apache.org/GMOxDOC11/apache-geronimo-v11-users-guide.html, if you want to know more about web application in Geronimo. When you unpack the sources (found in Section #What is needed) and follow the included Readme file you can deploy a web application consisting of 3 simple servlets (ServletStatistic, ServeletData, ServletSystemTime). Now you should be able to call the servlets. For example: http://localhost:8080/mysample/app1/ServletSystemTime.

Server-Side Monitor Component

The server-side monitor component consists of 2 classes (see Fig. #Class diagram of the management component example):

- the \verb+ServletMonitorGBean+, which implements the `GBeanLifecyle` interface and
- the \verb+AverageBuilder+ thread, which is doing the actual collection of data from the servlet MBeans. If the calculated average processing time is above the threshold an alarm is generated. The alarming function is not sending an e-mail, it is only logging "ALARM" to keep the code simple.



The Monitor GBean

The Geronimo framework is injecting information about the MBean server reference and the MBean servlet object names into the constructor during deployment. Listing {#Listing 1] shows you how to inform the Geronimo kernel about the ability of the GBean. The `GBeanInfoBuilder` is used to tell Geronimo the name of the GBean class. Then the constructor injection is defined by `addAttribute` to specify the three servlet MBean names, `addRefer ence` to specify the MBean server reference, and `setConstructor` to specify the constructor call.

ServletMonitorGBean.java (cutout)

```
public static final GBeanInfo GBEAN_INFO;
static {
 GBeanInfoBuilder infoB = GBeanInfoBuilder.createStatic("ServletMonitorGBean", ServletMonitorGBean.class);
 infoB.addAttribute("servletName1", String.class, true);
 infoB.addAttribute("servletName2", String.class, true);
 infoB.addAttribute("servletName3", String.class, true);
 infoB.addReference("MBeanServerReference", MBeanServerReference.class);
 infoB.setConstructor(new String[] { "MBeanServerReference", "servletName1", "servletName2", "servletName3"});
```

Further we want the methods, `startCalcAverage`, `stopCalcAverage`, `defineThreshold` and `defineUpdateInterval` to be available by an JMX management client. This allows us to interact with the monitor component through a MBean using the JConsole as shown in Section #Accessing Geronimo using JConsole. Publishing GBean methods through a MBean is done by the `addOperation` of the `GBeanInfoBuilder` by specifying the method names and parameters, as seen in Listing #Listing 2:

ServletMonitorGBean.java (cutout)

```
infoB.addOperation("startCalcAverage");
 infoB.addOperation("stopCalcAverage");
 infoB.addOperation("defineThreshold",new Class[]{int.class});
 infoB.addOperation("defineUpdateInterval",new Class[]{int.class});
```

The MBean configuration process is finished by setting the variable `GBEAN_INFO`. Thus the `MBeanServerKernelBridge` can use `getGBeanInfo()` to get a `GBeanInfo` object (Listing #Listing 3) with the MBean configuration and builds the appropriate MBean MB_{MC} Figure: #Geronimo architecture with integrated monitor component.

ServletMonitorGBean.java (cutout)

```
GBEAN_INFO = infoB.getBeanInfo();
public static GBeanInfo getGBeanInfo() { return GBEAN_INFO; }
```

The AverageBuilder Thread

The `AverageBuilder` class (Fig. #Class diagram of the management component example) is a thread which is querying the servlet MBeans. To query information from servlet MBeans, like the `requestCounter`, you have to know the object name of the MBean. You can find out about the object name using a JMX management client. In the section, #Interacting with the Monitor Component, it is shown how to find out about the object names of the servlet MBeans. Since we injected this information during the deployment process of the GBean, we are not to worry about that at the moment. Having the MBean object name we get the information `precessingTime` and `requestCount` as follows (see Listing #Listing 4):

ServletMonitorGBean.java (cutout)

```
ObjectName objNameStatisticServlet = ObjectName.getInstance(statisticServlet);
ObjectName objNameDataServlet=ObjectName.getInstance(dataServlet);
ObjectName objNameSystemTimeServlet = ObjectName.getInstance(systemTimeServlet);
long pt1=((Long)mbserver.getAttribute(objNameStatisticServlet, "processingTime")).longValue();
int rc1=((Integer)mbserver.getAttribute(objNameStatisticServlet, "requestCount")).intValue();
```

Doing this for all 3 servlets and building the average can be seen in the source code.

Deploying the Server-Side Monitor Component

When the GBean monitor component is compiled, it has to be deployed into Geronimo.
For deploying the monitor component everything has to be packaged according to Geronimo (see Geronimo's User Guide http://cwiki.apache.org/GMOxDOC11/apache-geronimo-v11-users-guide.html). That means you have to pack your GBeans with a deployment-plan into a jar file, e.g. `SimpleServletMonitor-1.0.jar` and deploy it. For the deployment, we have chosen the java deployment program, deployer:

```
java -jar $GERONIMOHOME/bin/deployer.jar --user system --password manager deploy SimpleServletMonitor-1.0.jar
```

It is a lot faster using the deployer program, then doing all the clicking using Geronimo's browser based management tool (http://localhost:8080/console).
With the example code, there comes also some simple Linux shell scripts supporting deployment, un-deployment, listing, etc.

Deployment Plans for Geronimo's GBeans

Geronimo's deployment plans (`geronimo-service.xml`) state information about GBean names, references and dependencies. For a more detailed description about the deployment descriptor, see http://cwiki.apache.org/GMOxDOC11/apache-geronimo-v11-users-guide.html. Our deployment descriptor (Listing #Listing 5) consists of an `<environment>` and and `<gbean>` part:

geronimo-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="http://geronimo.apache.org/xml/ns/deployment-1.1">
  <environment>...</environment>
  <gbean ...>...</gbean>
</module>
```

The ``environment'' part:

We have to give our monitor GBean an unique name inside Geronimo by defining its `groupId`, `artifactId`, `version` and `type`. Furthermore we have to define `<dependencies>` (see Listing: #Listing6). Because we want to inject the object reference for `MBeanServerReference` to the GBean constructor our deployment process depends on the `geronimo/rmi-naming` defined as the artifactId in the `geronimo-service.xml`.
The serialized version of that reference is found in the repository under: `geronimo/rmi-naming`.

geronimo-service.xml

```
<environment>
  <moduleId>
    <groupId>mysample</groupId>
    <artifactId>SimpleServletMonitor</artifactId>
    <version>1.0</version>
    <type>car</type>
  </moduleId>
  <dependencies>
    <dependency>
      <groupId>geronimo</groupId>
      <artifactId>rmi-naming</artifactId>
      <type>car</type>
    </dependency>
  </dependencies>
</environment>
```

The ``gbean'' part:

First you have to define the name of the GBean and the exact class name as XML attributes of the element <gbean> (see Listing: #Listing7).
Then you define all the references and attributes, which should be injected during deployment.

geronimo-service.xml

```
<gbean name="ServletMonitorGBean" class="de.hsfurtwangen.informatik.ServletMonitorGBean">
  <reference name="MBeanServerReference">
    <name>MBeanServerReference</name>
  </reference>
  <attribute name="servletName1" type="java.lang.String">
    Geronimo:j2eeType=Servlet,name=ServletStatistic,WebModule=//localhost/mysample,J2EEApplication=none,
J2EEServer=none
  </attribute>
  <attribute name="servletName2" type="java.lang.String">
    Geronimo:j2eeType=Servlet,name=ServletData,WebModule=//localhost/mysample,J2EEApplication=none,
J2EEServer=none
  </attribute>
  <attribute name="servletName3" type="java.lang.String">
    Geronimo:j2eeType=Servlet,name=ServletSystemTime,WebModule=//localhost/mysample,J2EEApplication=none,
J2EEServer=none
  </attribute>
</gbean>
```

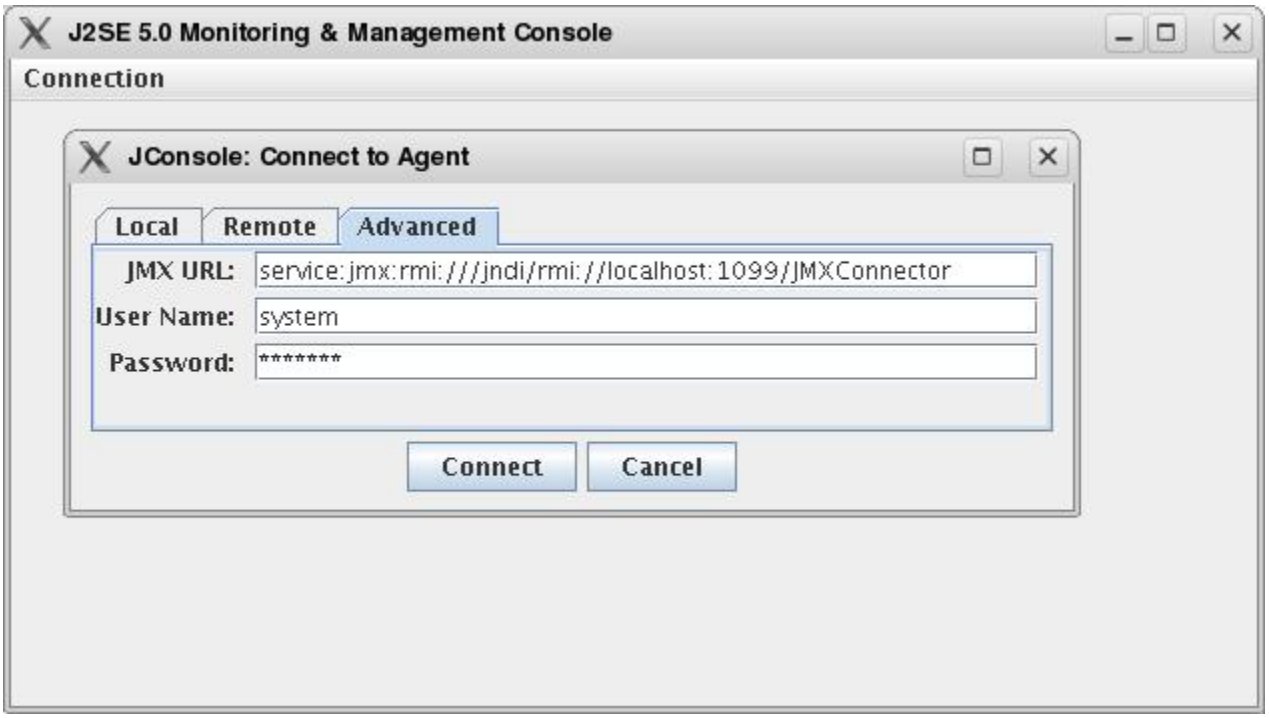
As seen in our example above (Listing: #Listing7), the following parameters are injected by constructor injection of the Geronimo framework:

- MBeanServerReference: The corresponding source code in the GBean class is: `GBeanInfoBuilder.addReference ("MBeanServerReference",MBeanServerReference.class)`
- servleteName1: The corresponding source code in the GBean class is: `GBeanInfoBuilder.addAttribute("servletName1",String. class)`
- servletName2: The corresponding source code in the GBean class is: `GBeanInfoBuilder.addAttribute("servletName2",String. class)`
- servletName3: The corresponding source code in the GBean class is: `GBeanInfoBuilder.addAttribute("servletName3",String. class)`

Testing the Monitor Component

Accessing Geronimo using JConsole

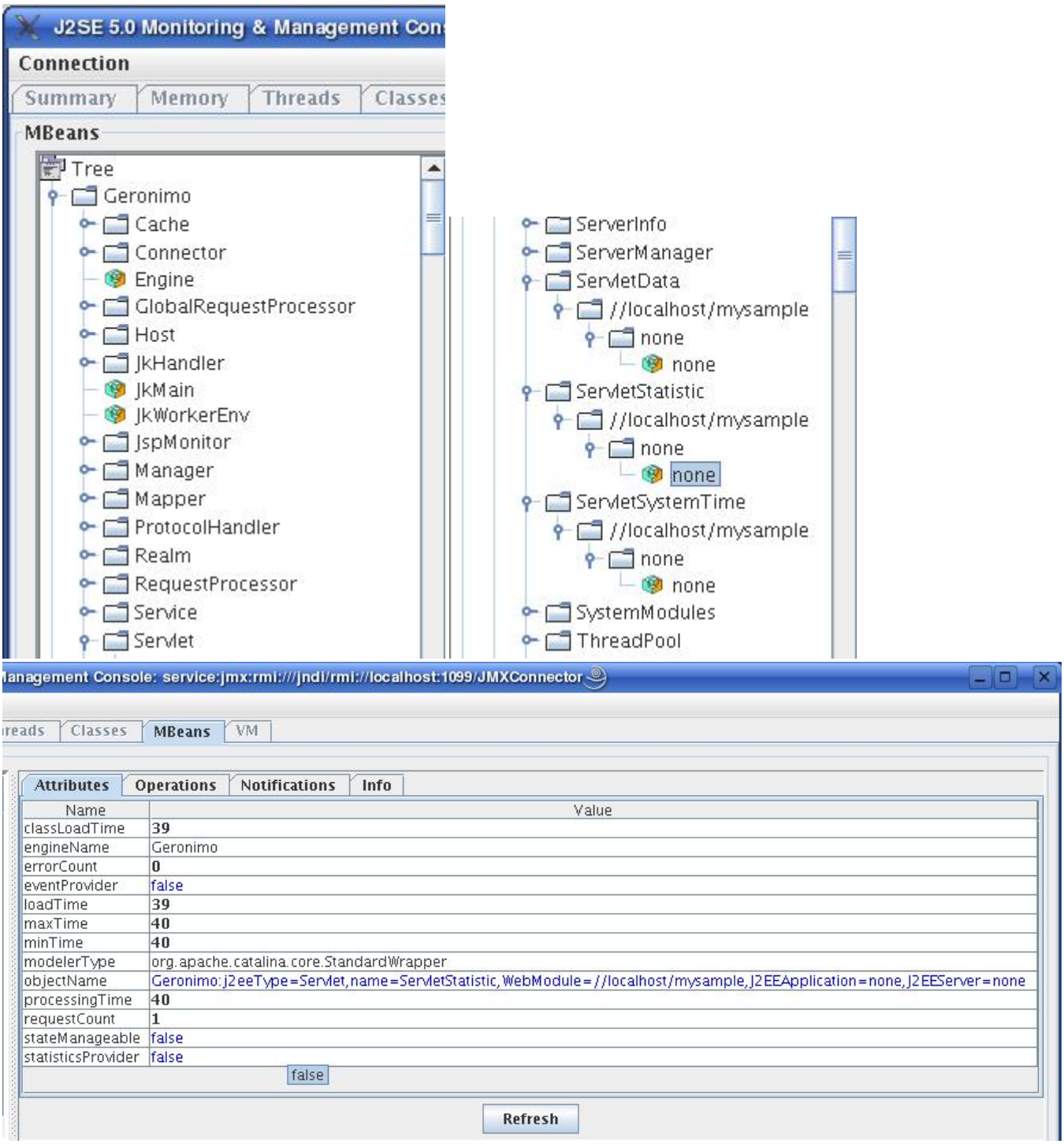
We want to use JConsole http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html, http://java.sun.com/j2se/1.5.0/docs/tooldocs/share /jconsole.html, which is a JMX-compliant GUI tool that connects to a running JVM. If the `JAVA_HOME` environment variable is set to a JDK 5.0, you only have to type in:



To connect to your Geronimo server you have to use the following settings in your JConsole (see Fig. #JConsole):

- **JMX URL:** `service:jmx:rmi:///jndi/rmi://localhost:1099/JMXConnector`
- **User Name:** `system`
- **Password:** `manager`

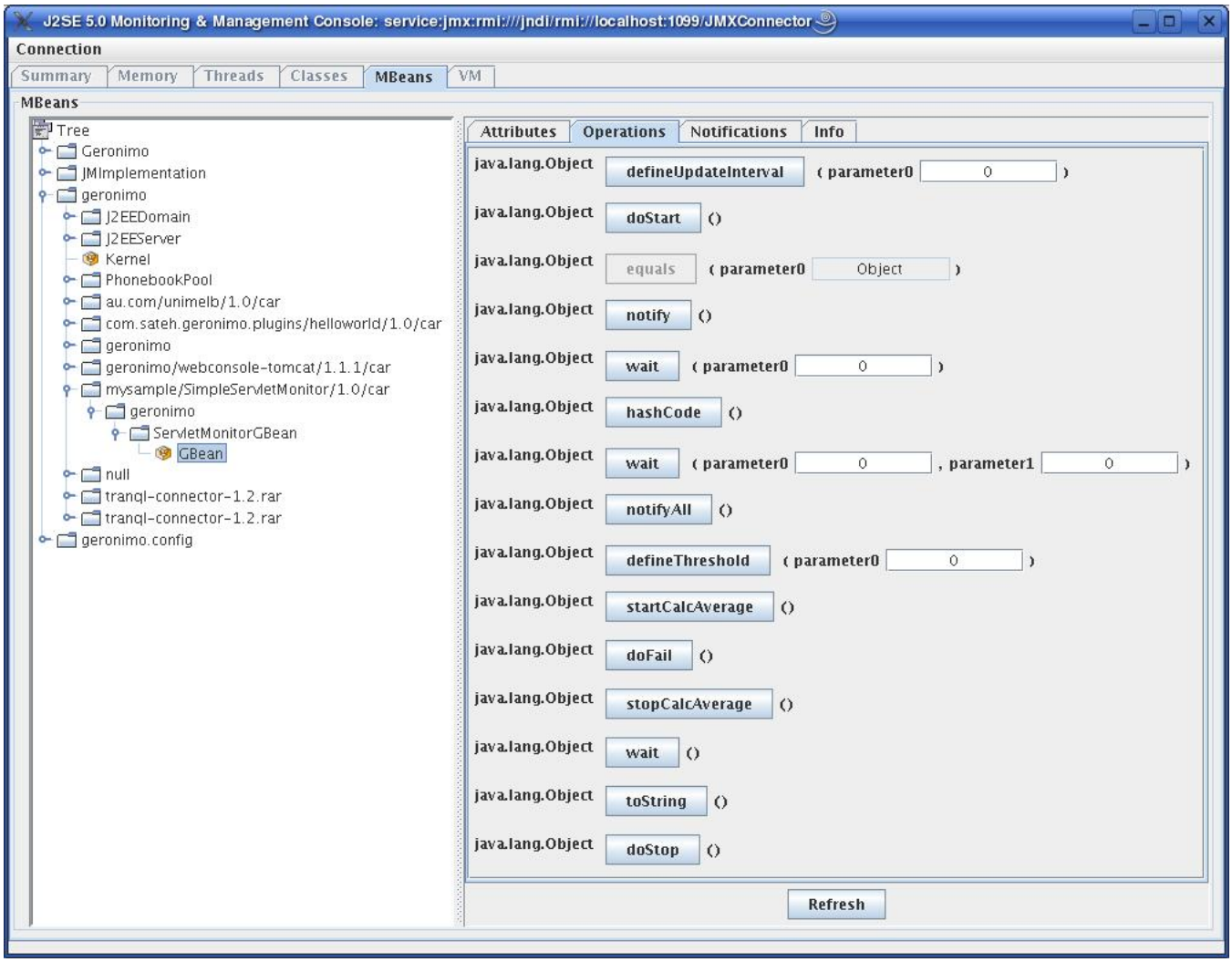
After the management client is connected to Geronimo you switch to the MBean view (see Fig. #Servlet MBean view) and navigate to one of the deployed servlets, e.g. `ServletStatistic` (1st and 2nd figure of Fig. #Servlet MBean view).



One interesting information is the object name (3rd figure of Fig. #Servlet MBean view)) that is used in the deployment descriptor for Geronimo's GBean deployment plan (see section #Deployment Plans for Geronimo's GBeans).

Interacting with the Monitor Component: ServletMonitorGBean

First, navigate to the GBean (see Fig. #GBean Operations). Then you switch to the operations view. Now you have access to all the methods you defined in the GBean with \verb+GBeanInfoBuilder+. Let's start the process by calculating the average processing time of the 3 servlets. Just click on \verb+startCalcAverage()+ and you can see in the \verb+GERONIMO_HOME/var/log/geronimo.out+ file that the averages are calculated. Now reload the servlets several times, so that you can see changes in the output. You could also change the update time for the information polling of the servlets. Click first on \verb+parameter0+ and set the value to 5000 and then on \verb+defineUpdateInterval+. Now every 5s the data is collected from the servlets. Play around and have fun.



Conclusion

An overview of the used technologies, Geronimo's GBeans and JMX has been given. You should now have an idea how to develop and deploy a monitor component inside Geronimo application server and how to improve the monitoring. The developed example shows a server's ability to do its own monitoring and application specific alarming. There are a lot more possibilities what can be done. For example a server could monitor the number of servlet requests and if there are more then, e.g. 20 per seconds, there might be a denial of service attack. Analyze the monitored data inside the server and optimize Geronimo's configuration, e.g. thread pool size. etc.

References

- Sun's java management extensions (JMX) page. http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/.
- JConsole reference. http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jconsole.html.
- JConsole manual. http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html.
- JSR-77: J2EE management specification. http://jcp.org/en/jsr/detail?id=77.
- Apache home page. http://www.apache.org/.
- Tomcat home page. http://tomcat.apache.org/.
- OpenEJB home page. http://incubator.apache.org/openejb/.
- Martin Fowler; *Inversion of control containers and the dependency injection pattern.*; http://www.martinfowler.com/articles/injection.html; January; 2004.
- Niel Frod; *Dependency injection in apache geronimo, part 1: A new way to look at decoupling in j2ee applications.*; http://www-128.ibm.com/developerworks/opensource/library/os-ag-ioc1/; February; 2006.
- Niel Frod; *Dependency injection in apache geronimo, part 2: The next generation.*; http://www-128.ibm.com/developerworks/opensource/library/os-ag-ioc2/; February; 2006.