Oracle® JRockit Mission Control

Introduction to Oracle JRockit Mission Control

3.0.3

June 2008

ORACLE

Oracle JRockit Mission Control Introduction to Oracle JRockit Mission Control, 3.0.3

Contents

Introduction to Oracle JRockit Mission Control

JRockit Mission Control Communications

Integration with the Eclipse IDE

Overview of JRockit Mission Control 3.0

Overview of JRockit Mission Control 2.0

Overview of Oracle JRockit 1.0

CHAPTER 1

Introduction to Oracle JRockit Mission Control

The Oracle JRockit Mission Control tools suite was introduced with Oracle JRockit JVM R26.0.0. It includes tools to monitor, manage, profile, and eliminate memory leaks in your Java application without introducing the performance overhead normally associated with these types of tools.

The JRockit Mission Control's low performance overhead is a result of using data collected as part of the JRockit JVM's normal adaptive dynamic optimization. This also eliminates the problem with the Heisenberg anomaly that can occur when tools using byte code instrumentation alters the execution characteristics of the system. The JRockit Mission Control functionality can always be available on-demand and the small performance overhead is only in effect while the tools are running.

This section contains information on the following subjects:

Installation Information

Installation instructions can be found in *Installing* Oracle JRockit Mission Control.

Starting the JRockit Mission Control Client

The JRockit Mission Control Client executable is located in `JROCKIT_HOME/bin`. If this directory is on your system path, you can start JRockit Mission Control by simply typing `jrmc` in a command (shell) prompt.

Otherwise, you have to type the full path to the executable file, as shown below:

```
JROCKIT_HOME\bin\jrmc.exe (Windows)
JROCKIT_HOME/bin/jrmc (Linux)
```

On Windows installations, you can start JRockit Mission Control from the **Start** menu.

JRockit Mission Control FAQ

This topic lists and provides answers to questions frequently asked about JRockit Mission Control.

I can not connect the JRockit Mission Control Client. What could be the problem?

Consider the following:

- Are you using the correct protocol?

  The easiest way is to ensure that you are using the same version of the JRockit JVM you want to monitor as the JVM running the JRockit Mission Control client. If that is not an option, you can use the radio buttons in the connection dialog box in the JRockit Mission

Control Client to select which protocol to use: 1.4 will select RMP and 1.5 and later will select JMXRMI.

For earlier versions of JRockit Mission Control these radio buttons don't exist and, to make a JRockit JVM 5.0 instance connect to a 1.4 version, you must explicitly specify the JMX Service URL. The format of the service URL is:

`service:jmx:rmp://<hostname>:<port>`

for example:

`service:jmx:rmp://localhost:7091`

- Are the correct ports opened?

  JMX over RMI uses two ports and that one of the ports will not be known beforehand.

- Is the communication caught in the firewall?

  Please see JRockit Mission Control Communications for more information.

When attempting to connect to JRockit Mission Control I get a stack trace indicating that JRockit Mission Control attempts to communicate with a strange IP or host name.

Sometimes RMI can have a problem determining which address to use. This can happen because of

- Access restrictions in the Security manager.
- The machine being multihomed and RMI picking the wrong interface.
- A misconfigured hosts file or a number of different network related configuration problems.

If all else fails you can try specifying the java.rmi.server.hostname system property. Please note that this can affect applications running in the JVM.

I'm getting exceptions during startup about classes not being found

Make sure you are using the proper launcher to start up the JRockit Mission Control Client. You must only use `JROCKIT_HOME/bin/jrmc`.

JRockit Mission Control can't find any local JVMs

Make sure you are using the proper launcher to start up the JRockit Mission Control Client. You must only use `JROCKIT_HOME/bin/jrmc`.

Why can't I see any Method Profiling information in my JRA recording?

By default, the JRockit Mission Control Client doesn't show tabs if no data has been recorded for them. Ensure that method profiling was enabled for your JRA recording and that the application was under load. If the JRockit JVM is spending most of the time with none of the threads doing any work, no samples will be recorded. If you still want to create a JRA recording with method sampling and a low load, try increasing the sampling frequency.

When using the Memory Leak Detector, nothing happens in the growth column of the trend table

The algorithm needs at least three data points to kick in and the data is collected as part of the old space mark phase of the garbage collection. If you see no data, possibly not enough garbage has been collected for these collections to occur. To speed up the process, try clicking the garbage can in the toolbar of the Memory Leak application to force three successive garbage collections, with a brief pause in between each collection.

JRockit Mission Control Documentation

Documentation for JRockit Mission Control 1 is available on eDocs:

http://edocs.bea.com/jrockit/tools/index.html

Documentation for JRockit Mission Control 2 and 3 is available both as online help with the installation of the tool and as pdfs on eDocs:

http://edocs.bea.com/jrockit/tools/index.html

Note: The pdfs for JRockit Mission Control 2 on eDocs are not the full versions. Look in the Help menu for full documentation.

JRockit Mission Control Support

You are entitled to support if you have an Enterprise licence.

Is There a Forum Where I can Discuss JRockit Mission Control?

If you have any questions you are welcome to share them in the Oracle JRockit general interest news group, which is monitored by the Oracle JRockit engineering team. To access the news group, go to:

http://newsgroups.bea.com

Giving Feedback the JRockit Mission Control Development Team

If you have any suggestions about how to improve the JRockit Mission Control plug-ins or information on how it is most commonly used in your development environments, we would be grateful to receive your input. This information would contribute to our understanding on how to best further improve these tools in the future.

Please, send an e-mail with feedback and your ideas on how to use it to:

jrockit-improve@bea.com

The feedback will be considered by the development team designing the Oracle JRockit Mission Control plug-ins. We will look at collected ideas and improve the plug-ins to make them even easier to use. Oracle's goal with these plug-ins is to simplify the tasks in getting your applications to run as smoothly as possible on the Oracle JRockit JVM.

CHAPTER 2

JRockit Mission Control Communications

Depending upon which J2SE version on which you are running the Oracle JRockit JVM, certain aspects of the communications protocols will differ. This chapter describes the protocols and their differences resultant from the different J2SE version.

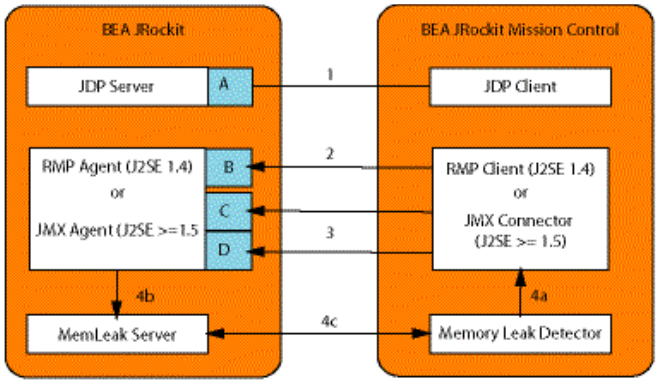
This section includes information on the following subjects:

- JRockit Mission Control Communications Overview
- J2SE 1.4
- J2SE 5.0 and Later
- All Versions

JRockit Mission Control Communications Overview

Figure 2-1 illustrates the communication topology for Oracle JRockit Mission Control.

Figure 2-1 JRockit Mission Control Communications Topology



1 Both versions announce their presence to JMC by using JDP when `-Xmanagement:autodiscovery=true`.

2 JMC communicates with J2SE 1.4 versions of JRockit over RMP through a port (B) configurable by using `-Xmanagement:port=#`.

3 JMC communicates with >=J2SE 5.0 versions of JRockit over JMXRMI. JMC connects to the *RMI registry* (C) through a port configurable by using `-Xmanagement:port=#` (E) while it connects to the the *RMI server* (D) through an anonymous, non-configurable port (F).

The Memory Leak Detector component of JMC uses the Memory Leak Protocol (MLP) to communicate with the JVM:

4a The RMP Client (1.4) or the JMX connector in JLMEXT (>=5.0) connects through the port (G) to the corresponding RMP server or JMX agent.

4b The RMP server (1.4) or JMX connector (>=5.0) receives the instructions to start the MemLeak server, which is native to the JVM.

4c In the Memory Leak Detector, the MLS Java API communicates with the MemLeak server.

J2SE 1.4

J2SE 1.4 versions of JRockit Mission Control uses RMP (Rockit Management Protocol.), an older legacy protocol that has existed since the 1.3 versions of the Oracle JRockit JDK. RMP uses a single socket. You can specify the port of the listening socket by using the `-Xmanagement`:port option; for example `-Xmanagement:port=7090`. Table 1-1 lists additional system properties you can use to further configure the agent.

Table 2-1 Additional Communication Settings for JRockit Mission Control on J2SE 1.4

System Property	Description	Default
`jrockit.managementserver.address`	Bind to a specific interface	Not enabled, listens on all interfaces)

Table 2-1 Additional Communication Settings for JRockit Mission Control on J2SE 1.4

System Property	Description	Default
`jrockit.managementserver.timeout`	Socket time-out in MS	4000
`jrockit.managementserver.maxconnect`	Maximum number of connections	4

J2SE 5.0 and Later

J2SE 5.0 and later versions of the JRockit JDK use JMXRMI (JMX over RMI). This protocol uses one port for the RMI registry, which is configured with the `-Xmanagement:port` option, and a second port (on an anonymous port) for communication with the RMI server. Note that you cannot configure the port for the RMI server; however, you can write your own agent that defines a fixed port for the RMI server. Please see the following link for further information:

`http://java.sun.com/javase/6/docs/technotes/guides/management/agent.html#gdfvv`

Table 2-2 lists the options available for the `-Xmanagement` flag:

Table 2-2 -Xmanagement Option

Option	Description	Default
`authenticate`	Use password authentication	True
`ssl`	Use secure sockets layer	True
`port`	What port to use for the RMI registry	7091

For a more comprehensive discussion on what these options mean, please see:

`http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html`.

All Versions

For all J2SE versions, you can use the `-Xmanagement` option autodiscovery to make the JRockit JVM use the JRockit Discovery Protocol (JDP) to announce its presence; for example `-Xmanagement:autodiscovery=true`.

Table 2-3 lists additional system properties you can use to control the behavior of the JDP server:

Table 2-3 System Properties Used to Control the JDP Server

System property	Description	Default
`jrockit.managementserver.discovery.period`	The time to wait between multicasting the presence in ms	5000
`jrockit.managementserver.discovery.ttl`	The number of router hops the packets being multicasted should survive	1
`jrockit.managementserver.discovery.address`	The multicast group/address to use	232.192.1.212
`jrockit.managementserver.discovery.targetport`	The target port to broadcast	7090(1.4)/7091(1.5)

All versions of JRockit Mission Control also employ an additional protocol when using the Memory Leak Detector. The memleak server is not written in Java; rather it is an integral part of the JVM. This is because a potential use case for the memleak server is to optionally be able to start it when an out of memory condition occurs in the JVM. When such a condition occurs, it is impossible to execute Java code because no heap would be available.

MLP (MemLeak Protocol) is used by the native memleak server during a memleak session. JRockit Mission Control communicates over RMP (1.4) or JMXRMI (5.0 and higher) to ask the Oracle JRockit JVM to start up the server. You can configure the port on which you want to start the memleak server on, and to use for the session, by using Oracle JRockit Mission Control preferences. See Figure 2-1 for schematic of this process.

CHAPTER 3

Integration with the Eclipse IDE

In addition to the standalone Rich Client Platform (RCP) version of Oracle JRockit Mission Control 3.0.3, the toolset is also available as a plug-in to the Eclipse IDE (Eclipse 3.3 or above). This version of JRockit Mission Control provides seamless integration of JRockit Mission Control's application profiling and monitoring toolset with the Eclipse development platform. By integrating the JRockit Mission Control Client with Eclipse, you can combine the features of Eclipse with the power toolset in JRockit Mission Control.

This document describes this integration and provides instructions for using the special functionality enabled by integrating the JRockit Mission Control Client with Eclipse. The topics include:

Benefits of the Integration

When the JRockit Mission Control Client is run within the Eclipse IDE, you have access to IDE features that aren't otherwise available in the toolset when it is run as a standalone Rich Client Platform (RCP) application. The most significant of these features is the ability to see specific

code in the running application by opening it directly from the JRockit Mission Control Client, a function called Jump-to-Source.

The other obvious benefit of integrating the JRockit Mission Control Client with the Eclipse IDE is that it allows you to profile and monitor an application during their development phase just as you would during their production phase. This allows you to spot potential runtime problems before you actually deploy your application to production; for example, you might, while monitoring an application during its development notice a memory leak. By catching the memory leak during development, you can correct it before you migrate your application to a production environment.

Differences between the Eclipse Version and the RCP Version

Generally, the Eclipse version of the JRockit Mission Control Client works identically to the RCP version. Any component in the Eclipse version offers the same functionality and user interface as the comparable component delivered on the RCP.

The biggest difference that Eclipse version of the JRockit Mission Control Client has over the RCP version is the Jump-to-Source feature, described in Jumping to Application Source. With this feature, you can not only see the name of a “problem” class or method displayed in the JRockit Mission Control Client, but you can jump from the displayed name directly to that class or method’s source, where you can evaluate the code to see what might be causing the problem. Jump-to-Source is enabled for the Management Console, the JRockit Runtime Analyzer, and the Memory Leak Detector.

Making the JRockit JVM Your JVM

While JRockit Mission Control can work with many different Java Virtual Machines, it is highly recommended that you use the Oracle JRockit JVM as your JVM when running Mission Control on the Eclipse platform. Not only will you avail yourself of the JRockit JVM’s exceptional performance, but by using this JVM, Mission Control’s autodetect feature will be enabled, which makes it simple to connect Mission Control to your locally running application.

To run Eclipse (and thus the JRockit Mission Control Client) on the JRockit JVM

1. Go to your file system browser (for example, Windows Explorer).

2. Locate your Eclipse installation folder (for example, `C:\Program Files\Eclipse`) and, with a file editor other than Notepad, open the file `eclipse.ini`. It will look something like the example in Listing 3-1.

Listing 3-1 eclipse.ini Example

```
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256M
-vmargs
-Dosgi.requiredJavaVersion=1.5
-Xms40m
-Xmx512m
```

3. Make the following changes to eclipse.ini:
   - Remove all flags related to non-Oracle JRockit JVMs (for example, `--launcher.XXMaxPermSize 256M`)
   - On the third line down (after `org.eclipse.platform`), add the following:

     -vm
     <*Full path to* JRockit JVM*'s javaw file*>

     The full path to JRockit's javaw file might look like this on Windows:

     ```
     C:\Program Files\Java\jrockit-R27.4.0-jdk1.6.0_02\bin\javaw.exe
     ```

     or like this on Linux and Solaris:

     ```
     $HOME/jrockit-R27.4.0-jdk1.6.0_02/bin/javaw
     ```
   - Depending upon your particular JRockit JVM implementation and the applications running on it, you can set any valid JRockit JVM command-line option. For example, you might want to set a garbage collector that meets your system priorities by using the `-XgcPrio:` option or increase (or decrease) the initial and maximum heap size by changing the values for `-Xms` and `-Xmx`.

     For more information on tuning the JVM, please refer to Profiling and Performance Tuning in the Oracle JRockit JVM *Diagnostics Guide*.

For more information on the available command-line options, please refer to the Oracle JRockit JVM *Command-Line Reference*.

4. When you are done making the necessary changes to `eclipse.ini`, save and close the file. Listing 3-2 shows an example of the eclipse.ini file updated to make Oracle JRockit JVM the JVM.

Listing 3-2 Updated eclipse.ini file for a Windows implementation

```
-showsplash
org.eclipse.platform
-vm
C:\Program Files\Java\jrockit-R27.4.0-jdk1.6.0_02\bin\javaw.exe
-vmargs
-Dosgi.requiredJavaVersion=1.5
-Xms256m
-Xmx512m
-XgcPrio:pausetime
```

Selecting a Perspective

A “perspective” defines a set of views and their relative positions within the Eclipse window; in other words, it is a template for graphically presenting different types of information in Eclipse. For example, the Java perspective combines views that you would commonly use while editing Java source files, while the Debug perspective contains the views that you would use while debugging Java programs.

JRockit Mission Control plug-ins for Eclipse come with a predefined perspective called JRockit Mission Control. This perspective shows the JRockit Mission Control Client interface so that you can use the tools in JRockit Mission Control to profile applications as you develop them in Eclipse.

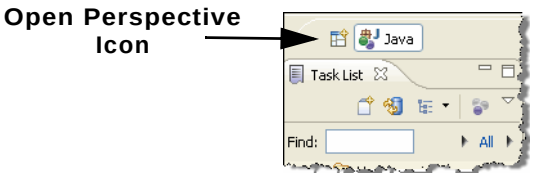
This topic will show you how:

- To open the Mission Control Perspective
- To change perspective from Mission Control
- To reopen the Mission Control Standard Perspective

To open the Mission Control Perspective

1. In top right corner of the Eclipse window, click the Open Perspective icon (Figure 3-1).

Figure 3-1 Open Perspective Icon



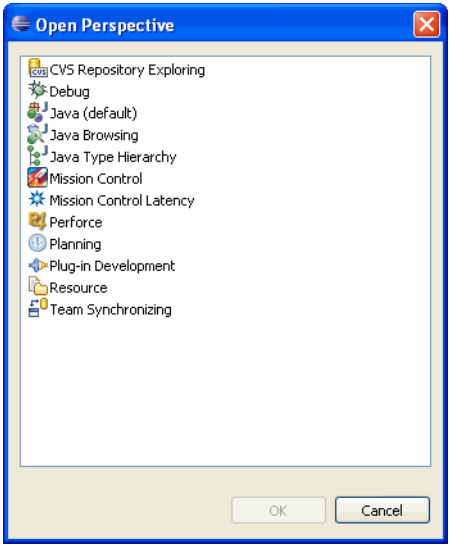
The Open Perspective context menu appears (Figure 3-2).

Figure 3-2 Open Perspective Context Menu

2. Select **Other...**

The Open Perspective dialog box appears (Figure 3-3).

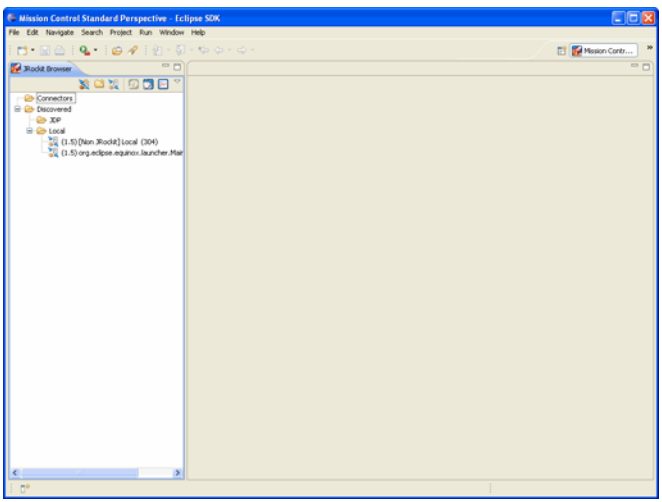
Figure 3-3 Open Perspective Dialog Box



3. Select **Mission Control** and click **OK**.

   The Eclipse window reconfigures to show the Mission Control Standard Perspective (Figure 3-4).

Figure 3-4 Mission Control Standard Perspective



To change perspective from Mission Control

You can change perspectives from Mission Control to another perspective by using one of the methods described in Table 3-1:

Table 3-1 Changing Perspectives

If...	Do this...
You've never opened the perspective	1. Click the Open Perspective icon. 2. Either: – Select the perspective you want to open. – If the perspective name does not appear on the context menu, select **Other...** to open the Open Perspective dialog box and select the perspective from there.
You've opened the perspective before	If you've opened the perspective before, a button for that perspective will appear in the top right corner of the Standard Mission Control Perspective, near the Open Perspective icon. Simply click the button for the perspective you want to open.

To reopen the Mission Control Standard Perspective

If you have already opened the Mission Control Standard Perspective for this project, a Mission Control button will appear next to the Open Perspective button in the top right corner of the Eclipse window (Figure 3-5).

Figure 3-5 Open Standard Mission Control Perspective Button

To reopen the perspective, simply click that button.

Jumping to Application Source

When running JRockit Mission Control plug-ins in an Eclipse IDE you can select a method or class and jump from the JRockit Mission Control Client directly to the source code where that method or class is declared. An editor will open up showing you the source file. Jump-to-Source is available in JRA, the Management Console and the Memory Leak Detector:

This topic contains the following information:

Using Jump-to-Source

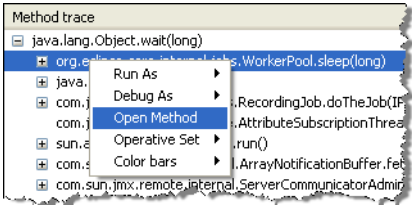
To jump from the JRockit Mission Control Client to source code

Note: The following procedure is generic. See JRockit Mission Control Plug-ins with Jump-to-Source Enabled for a list of plug-ins where this feature is enabled.

1. On the table, tree or other GUI component listing classes or methods, right-click the class or method for which you want to see the source code.

   A context menu appears (Figure 3-6).

Figure 3-6 Jump to Source Command on the Context Menu



2. Select **Open Method** (or **Open Type**, if you are jumping from to a class call).

   The associated source code will appear in a new editor.

JRockit Mission Control Plug-ins with Jump-to-Source Enabled

Note: This feature only works with versions of the JRockit Mission Control Client integrated into the Eclipse IDE.

Table 3-2 lists the Oracle JRockit Mission Control plug-ins where Jump-to-Source is enabled.

Table 3-2 Plug-ins with Jump-to-Source Enabled

Plug-in	Component
Management Console	• Threads tab – Stack traces for selected threads • Exception Counter – Profiling Information table
JRA	• Methods Tab – The tables and both the trees. • GCs Tab – The GC method call tree for a garbage collection. • GC General Tab – Garbage collection call trees. • Objects Tab – Both **Start of Recording** and **End of recording** • Optimizations: – In the table. • Locks – Java Locks • Latency Log – Event Details – Event Properties – Stack Trace • Latency Log – Event Property Histogram, • Latency Traces – The trace trees.
Memory Leak Detector	• Trend Table • Application Stack Traces

CHAPTER 4

Overview of JRockit Mission Control 3.0

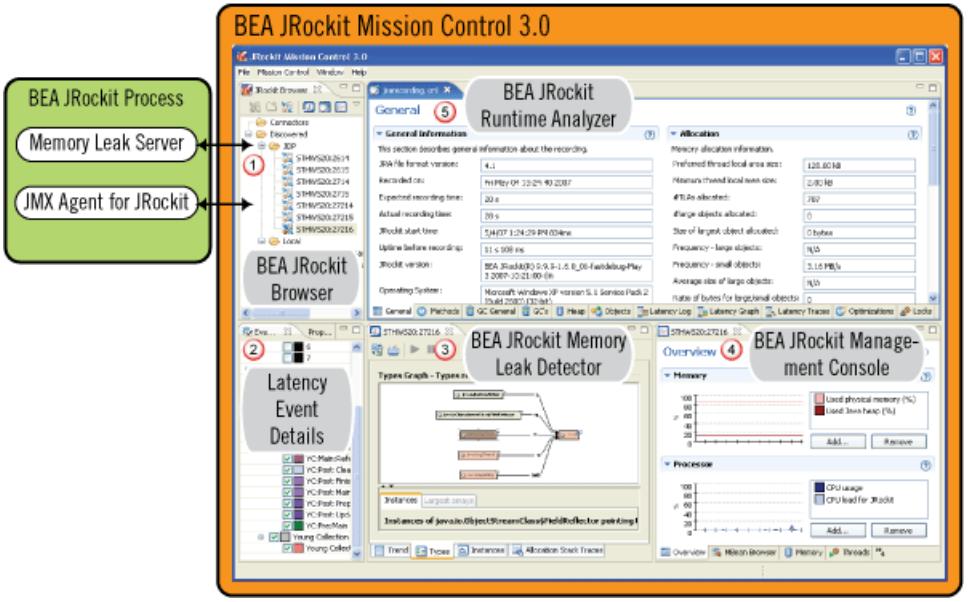
The Oracle JRockit Mission Control 3.0 tools suite includes tools to monitor, manage, profile, and eliminate memory leaks in your Java application without introducing the performance overhead normally associated with these types of tools.

This section contains information on the following subjects:

Architectural Overview of JRockit Mission Control 3.0

With the new Rich Client Platform (RCP) based JRockit Mission Control, you can launch the Memory Leak Detector, the JRockit Runtime Analyzer, and the JRockit Management Console from within the JRockit Mission Control. Figure 4-1 depicts how JRockit Mission Control looks when all tools are loaded.

Figure 4-1 Architectural Overview of JRockit Mission Control 3.0



When a JRA recording is started from within the JRockit Mission Control Client, it records the status of the JRockit JVM process for the time that you have specified and creates a ZIP file containing an XML file with the recorded data and optionally a binary file with latency data together with the corresponding data producer specification files. The ZIP file is automatically opened in the JRockit Runtime Analyzer tool upon completion of the recording, valid for JDK level 1.5 and later (marked 5 in Figure 4-1). Typical information that is recorded during a JRA recording is Java heap distribution, garbage collections, method samples, and lock profiling information (optional). New for the Oracle JRockit Mission Control 3.0 release, is that you can also record thread latency data. When viewing Latency data in the JRA Tool, the Latency Events Details become visible (marked 2 in Figure 4-1).

To view real-time behavior of your application and of Oracle JRockit JVM, you can connect to an instance of the JRockit JVM and view real-time information through the JRockit Management Console (marked 4 in Figure 4-1). Typical data that you can view is thread usage, CPU usage, and memory usage. All graphs are configurable and you can both add your own attributes and redefine their respective labels. In the Management Console you can also create rules that trigger on certain events, for example, an mail will be sent if the CPU reaches 90% of the size.

With the JMX Agent you have access to all MBeans deployed in the platform MBean server. From these MBeans, you can read attribute information, such as garbage collection pause times.

To find memory leaks in your Java application, you connect the JRockit Memory Leak Detector to the running JRockit JVM process. The Memory Leak Detector connects to the JMX (RMP) Agent that instructs to start a Memory Leak server where all further communication takes place.

Starting JRockit Mission Control

The JRockit Mission Control Client executable is located in `JROCKIT_HOME/bin`. If this directory is on your system path, you can start the JRockit Mission Control Client by simply typing `jrmc` in a command (shell) prompt.
Otherwise, you have to type the full path to the executable file, as shown below:

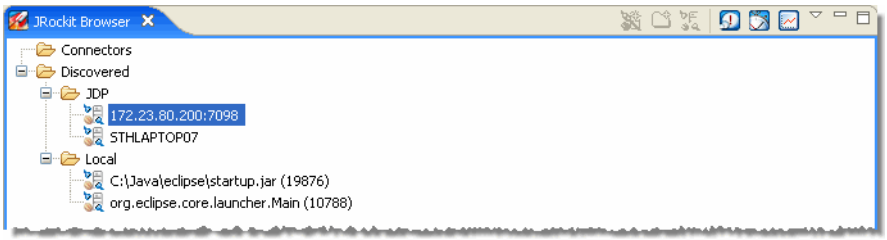
`JROCKIT_HOME\bin\jrmc.exe` (Windows)
`JROCKIT_HOME/bin/jrmc` (Linux)

On Windows installations, you can start JRockit Mission Control from the **Start** menu.

The JRockit Browser

The JRockit Browser (see Figure 4-2) was new for the JRockit Mission Control 2.0 release. This tool allows you to set up and manage all running instances of JRockit JVM on your system. From the JRockit Browser you activate different tools, such as starting a JRA recording, connecting a Management Console, and starting memory leak detection. Each JRockit JVM instance is referred to as a *Connector*.

Figure 4-2 The JRockit Browser

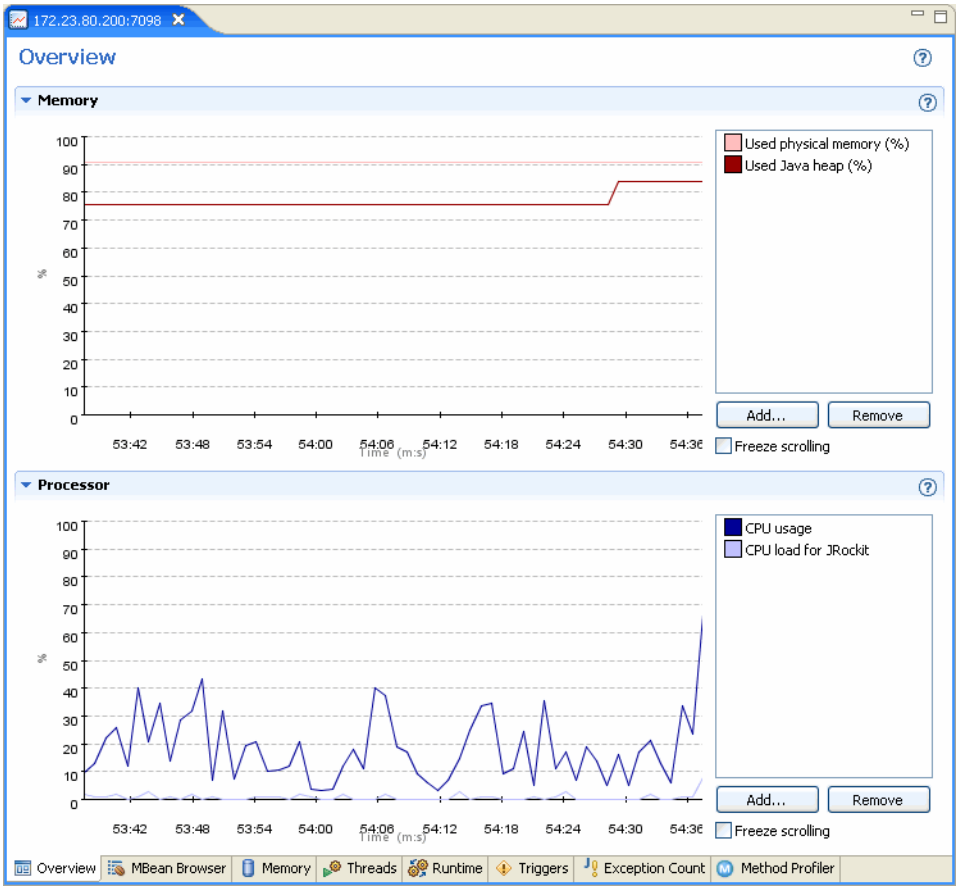


The JRockit Management Console

The JRockit Management Console (see Figure 4-3) is used to monitor a JRockit JVM instance. Several Management Consoles can be running concurrently side by side. The tool captures and presents live data about memory, CPU usage, and other runtime metrics. For the Management Console that is connected to JRockit JDK 5.0, information from any JMX MBean deployed in the

Oracle JRockit JVM internal MBean server can be displayed as well. For a Console connected to Oracle JRockit JDK 1.4, RMP capabilities are exposed by a JMX proxy. JVM management includes dynamic control over CPU affinity, garbage collection strategy, memory pool sizes, and more.

Figure 4-3 The JRockit Management Console

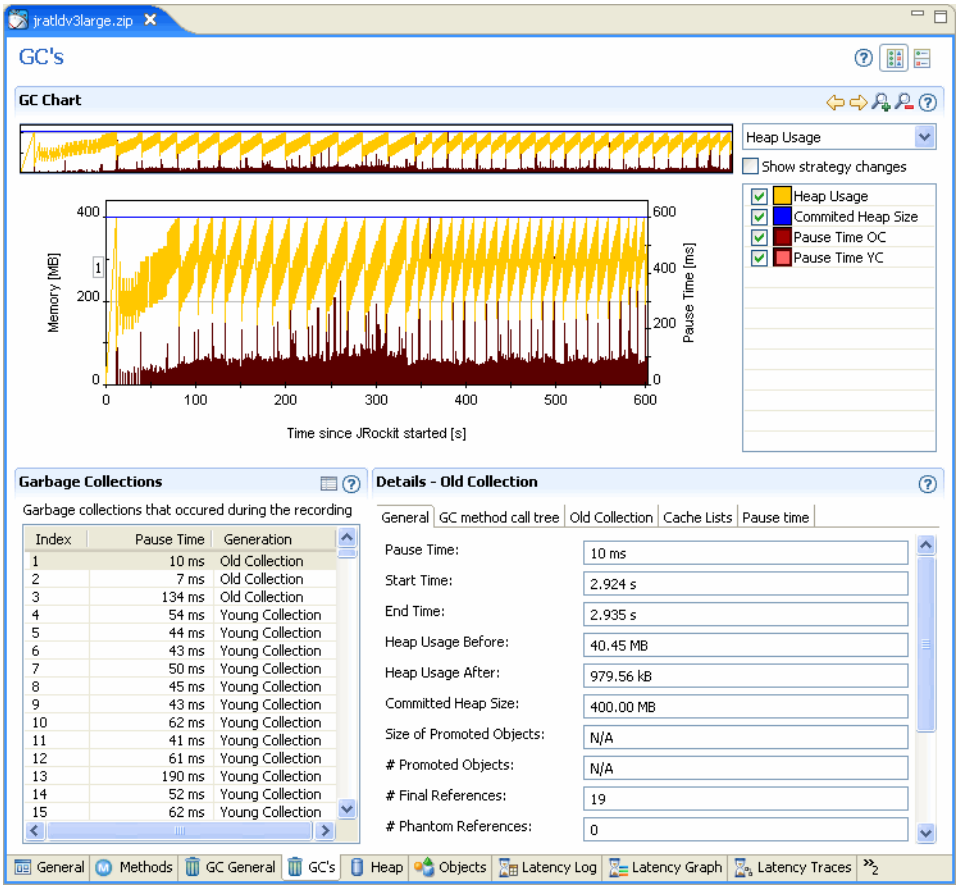


The JRockit Runtime Analyzer (JRA)

The JRockit Runtime Analyzer (see Figure 4-4) is an on-demand “flight recorder” that produces detailed recordings about the JVM and the application it is running. The recorded profile can later

be analyzed off line, using the JRA. Recorded data includes profiling of methods and locks, as well as garbage collection statistics, optimization decisions, and event latencies.

Figure 4-4 The JRockit Runtime Analyzer

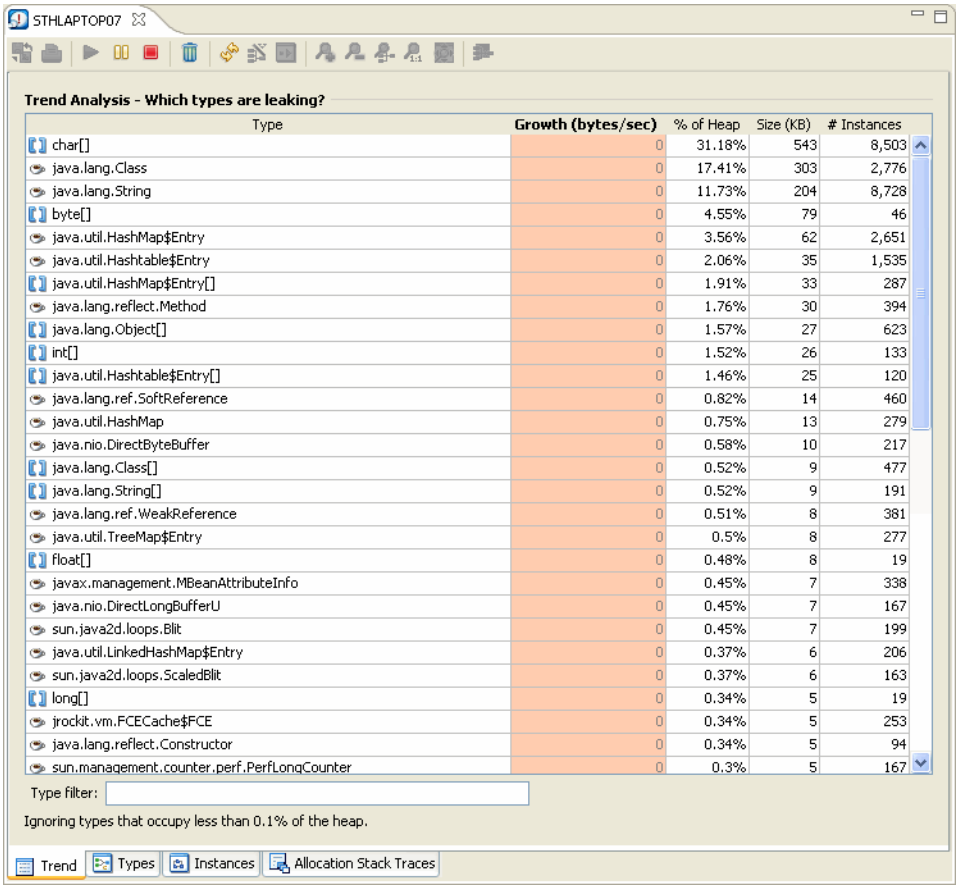


The JRockit Memory Leak Detector

The JRockit Memory Leak Detector (see Figure 4-5) is a tool for discovering and finding the cause for memory leaks in a Java application. The JRockit Memory Leak Detector's trend analyzer discovers slow leaks, it shows detailed heap statistics (including referring types and instances to leaking objects), allocation sites, and it provides a quick drill down to the cause of

the memory leak. The Memory Leak Detector uses advanced graphical presentation techniques to make it easier to navigate and understand the sometimes complex information.

Figure 4-5 The JRockit Memory Leak Detector



Type	Growth (bytes/sec)	% of Heap	Size (KB)	# Instances
char[]	0	31.18%	543	8,503
java.lang.Class	0	17.41%	303	2,776
java.lang.String	0	11.73%	204	8,728
byte[]	0	4.55%	79	46
java.util.HashMap$Entry	0	3.56%	62	2,651
java.util.Hashtable$Entry	0	2.06%	35	1,535
java.util.HashMap$Entry[]	0	1.91%	33	287
java.lang.reflect.Method	0	1.76%	30	394
java.lang.Object[]	0	1.57%	27	623
int[]	0	1.52%	26	133
java.util.Hashtable$Entry[]	0	1.46%	25	120
java.lang.ref.SoftReference	0	0.82%	14	460
java.util.HashMap	0	0.75%	13	279
java.nio.DirectByteBuffer	0	0.58%	10	217
java.lang.Class[]	0	0.52%	9	477
java.lang.String[]	0	0.52%	9	191
java.lang.ref.WeakReference	0	0.51%	8	381
java.util.TreeMap$Entry	0	0.5%	8	277
float[]	0	0.48%	8	19
javax.management.MBeanAttributeInfo	0	0.45%	7	338
java.nio.DirectLongBufferU	0	0.45%	7	167
sun.java2d.loops.Blit	0	0.45%	7	199
java.util.LinkedHashMap$Entry	0	0.37%	6	206
sun.java2d.loops.ScaledBlit	0	0.37%	6	163
long[]	0	0.34%	5	19
jrockit.vm.FCECache$FCE	0	0.34%	5	253
java.lang.reflect.Constructor	0	0.34%	5	94
sun.management.counter.perf.PerfLongCounter	0	0.3%	5	167

CHAPTER 5

Overview of JRockit Mission Control 2.0

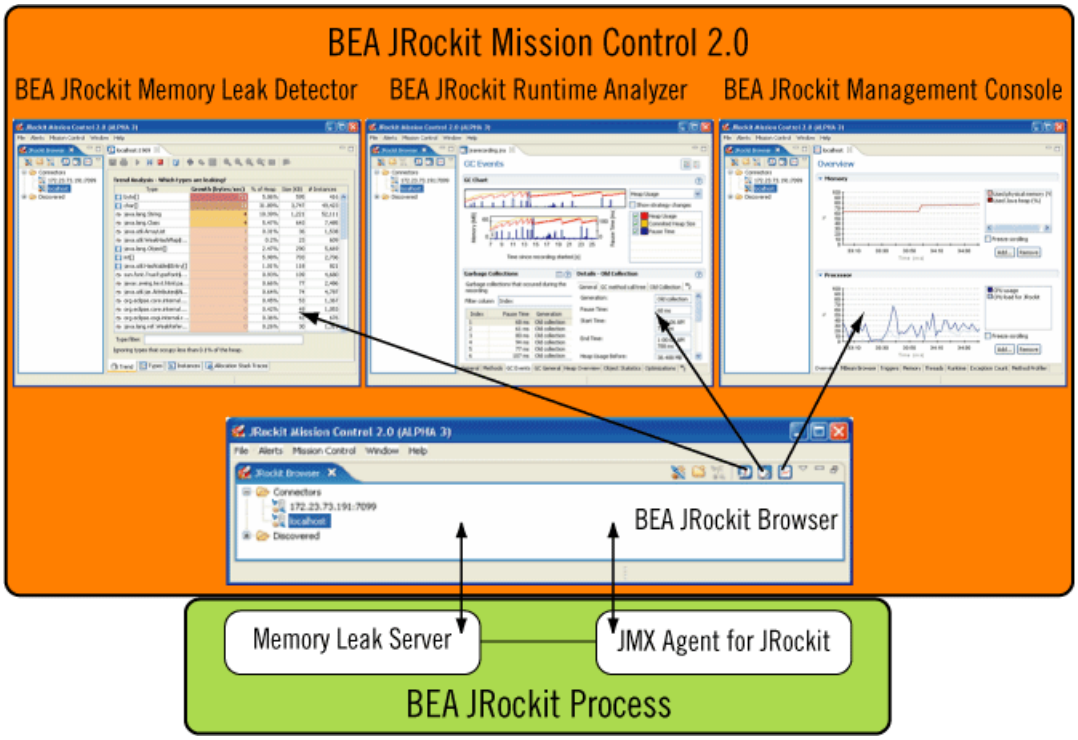
The Oracle JRockit Mission Control 2.0 tools suite includes tools to monitor, manage, profile, and eliminate memory leaks in your Java application without introducing the performance overhead normally associated with these types of tools.

This section contains information on the following subjects:

Architectural Overview of JRockit Mission Control 2.0

With the new Rich Client Platform (RCP) based JRockit Mission Control, you can launch the Memory Leak Detector, the JRockit Runtime Analyzer, and the JRockit Management Console from within the JRockit Mission Control (see Figure 5-1).

Figure 5-1 Architectural Overview of JRockit Mission Control 2.0



With the JMX Agent you have access to all MBeans deployed in the platform MBean server. From these MBeans, you can read attributes information, such as garbage collection pauses.

When a JRA recording is started from within JRockit Mission Control, it records the status of the JRockit JVM process for the time that you have specified and creates an XML file. This file is automatically opened in the JRockit Runtime Analyzer. Typical information that is recorded during a JRA recording is Java heap distribution, garbage collections, method optimizations, and method profiling information.

To find memory leaks in your Java application, you connect the JRockit Memory Leak Detector to the running JRockit JVM process. The Memory Leak Detector connects to the JMX (RMP) Agent that instructs to start a Memory Leak server where all further communication takes place.

Starting JRockit Mission Control

The JRockit Mission Control executable is located in `JROCKIT_HOME/bin`. If this directory is on your system path, you can start JRockit Mission Control by simply typing `jrmc` in a command (shell) prompt.
Otherwise, you have to type the full path to the executable file, as shown below:

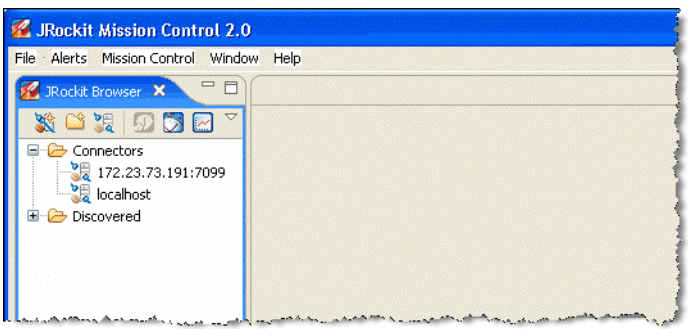
`JROCKIT_HOME/bin/jrmc.exe` (Windows)
`JROCKIT_HOME\bin\jrmc` (Linux)

On Windows installations, you can start JRockit Mission Control from the **Start** menu.

The JRockit Browser

The JRockit Browser (see Figure 5-2) is new for the JRockit Mission Control 2.0 release. This tool allows you to set up and manage all running instances of JRockit JVM on your system. From the JRockit Browser you activate recordings, set up a tree view of different JRockit JVM instances to monitor, start other JRockit Mission Control tools, etc. Each JRockit JVM instance is referred to as a Connector.

Figure 5-2 The JRockit Browser

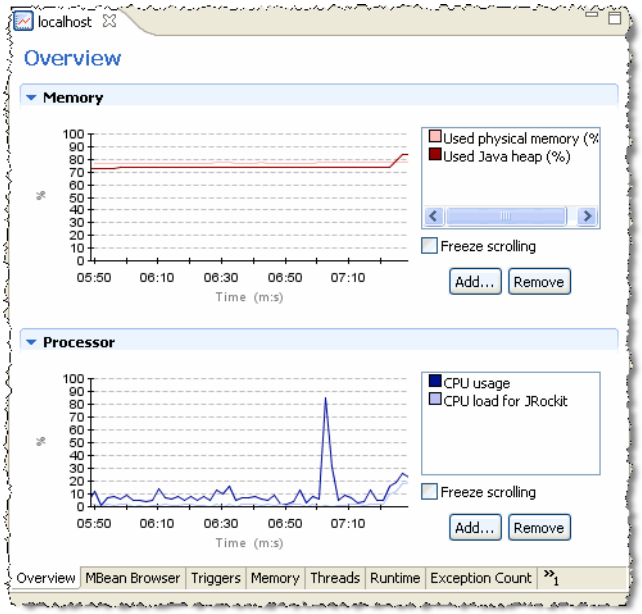


The JRockit Management Console

The JRockit Management Console (see Figure 5-3) is used to monitor a JRockit JVM instance. Several Management Consoles can be running concurrently side by side. The tool captures and presents live data about memory, CPU usage, and other runtime metrics. For the Management Console that is connected to Oracle JRockit JDK 5.0, information from any JMX MBean deployed in the JRockit JVM internal MBean server can be displayed as well. For a Console connected to Oracle JRockit JDK 1.4, RMP capabilities are exposed by a JMX proxy. JVM

management includes dynamic control over CPU affinity, garbage collection strategy, memory pool sizes, and more.

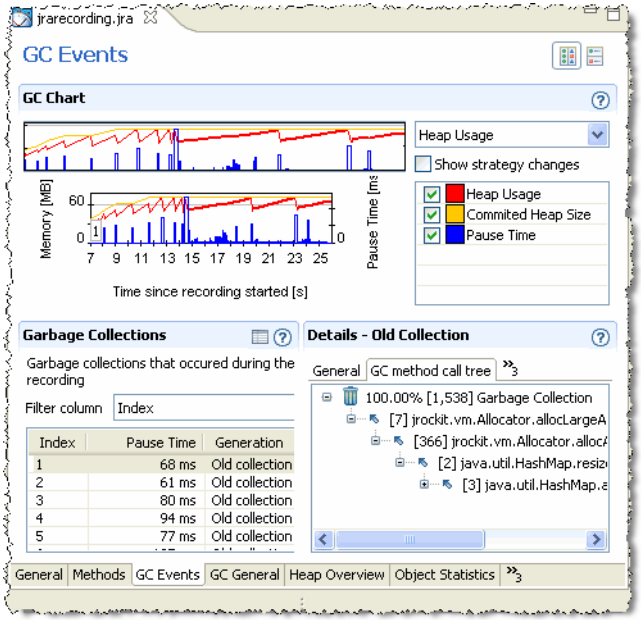
Figure 5-3 The JRockit Management Console



The JRockit Runtime Analyzer (JRA)

The JRockit Runtime Analyzer (see Figure 5-4) is an on-demand “flight recorder” that produces detailed recordings about the JVM and the application it is running. The recorded profile can later be analyzed off line, using the JRA tool. Recorded data includes profiling of methods and locks, as well as garbage collection statistics, optimization decisions, and object statistics.

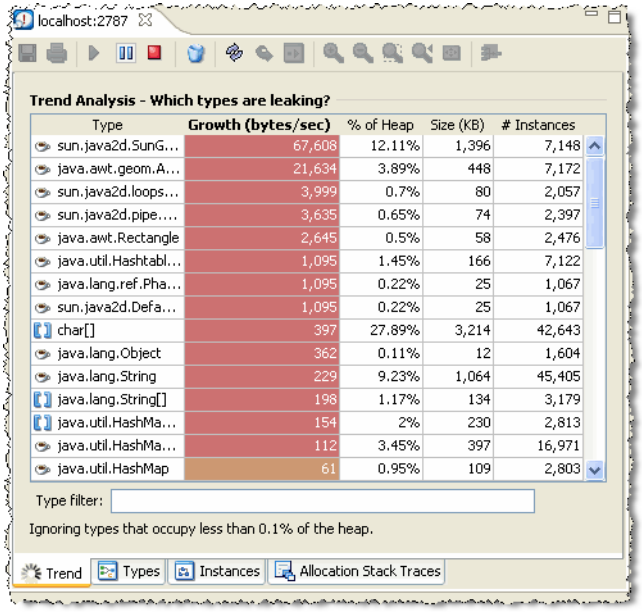
Figure 5-4 The JRockit Runtime Analyzer



The JRockit Memory Leak Detector

The JRockit Memory Leak Detector (see Figure 5-5) is a tool for discovering, and finding the cause for memory leaks in a Java application. The JRockit Memory Leak Detector's trend analyzer discovers slow leaks, it shows detailed heap statistics (including referring types and instances to leaking objects), allocation sites, and it provides a quick drill down to the cause of the memory leak. The Memory Leak Detector uses advanced graphical presentation techniques to make it easier to navigate and understand the sometimes complex information.

Figure 5-5 The JRockit Memory Leak Detector



CHAPTER 6

Overview of Oracle JRockit 1.0

The Oracle JRockit Mission Control 1.0 tools suite was introduced with Oracle JRockit JDK R26.0.0. These tools are run as standalone tools to the JRockit JVM.

This section contains information on the following subjects:

- Architectural Overview of JRockit Mission Control 1.0
- The JRockit Management Console
- The JRockit Runtime Analyzer (JRA)
- The JRockit Memory Leak Detector

Architectural Overview of JRockit Mission Control 1.0

JRockit Mission Control is available on Oracle JRockit JDK 1.4.2 (R26.2 and later) and JRockit JVM 5.0 (R26.0 and later), see Figure 1-1. The difference between the two is the connection agent used by the JRockit Management Console and the JRockit Management Console user interface itself.

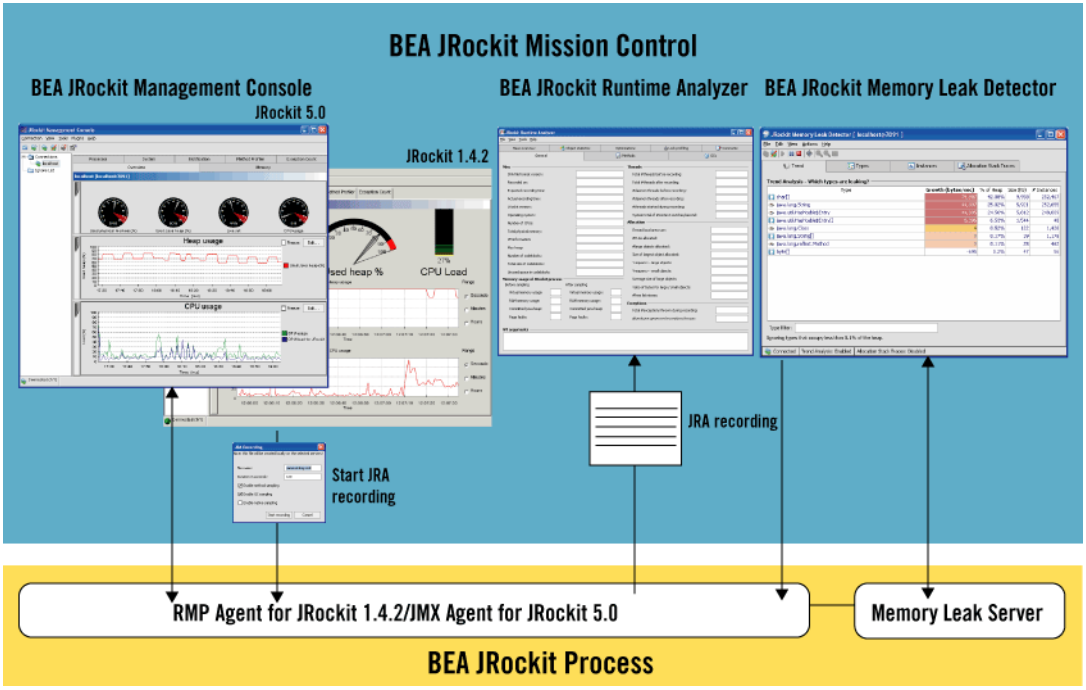
The RMP Agent (JRockit JDK 1.4.2) provides access, among other things, to live data about memory and CPU usage. With the addition of the JMX Agent (available with JRockit JDK 5.0) you will also get access to MBeans available to the platform MBean server. From these MBeans, you can read attributes information, such as garbage collection pauses.

When a JRA recording is started, for example, from the Management Console, it records the status of the JRockit JVM process for the time that you have specified. When the recording is completed, the information is saved to an XML file. This XML file can be viewed and analyzed

in the JRockit Runtime Analyzer tool. Typical information that is recorded during a JRA recording is Java heap distribution, garbage collections, and method optimizations.

To find memory leaks in your Java application, you connect the JRockit Memory Leak Detector to the running JRockit JVM process. The Memory Leak Detector connects to the JMX (RMP) Agent that instructs to start a Memory Leak server where all further communication takes place.

Figure 1-1 Architectural Overview of JRockit Mission Control 1.0



The JRockit Management Console

The JRockit Management Console is used to monitor and manage multiple (or single) JRockit JVM instances. It captures and presents live data about memory, CPU usage, and other runtime metrics. For the Management Console that is running on JRockit JDK 5.0, information from any JMX MBean deployed in the JRockit JVM internal MBean server (JMX Agent in Figure 1-1) can be displayed as well. JVM management includes dynamic control over CPU affinity, garbage collection strategy, memory pool sizes, and more.

The JRockit Runtime Analyzer (JRA)

The JRockit Runtime Analyzer (JRA) is an on-demand “flight recorder” that produces detailed recordings about the JVM and the application it is running. The recorded profile can later be analyzed off line, using the JRA tool. Recorded data includes profiling of methods and locks, as well as garbage collection statistics, optimization decisions, and object statistics.

The JRockit Memory Leak Detector

The JRockit Memory Leak Detector is a tool for discovering, and finding the cause for memory leaks in a Java application. The JRockit Memory Leak Detector’s trend analyzer discovers slow leaks, it shows detailed heap statistics (including referring types and instances to leaking objects), allocation sites, and it provides a quick drill down to the cause of the memory leak. The Memory Leak Detector uses advanced graphical presentation techniques to make it easier to navigate and understand the sometimes complex information.