ORACLE
FUSION MIDDLEWARE

An Oracle White Paper
May 2013

Java Usage Tracking and Visualization with Oracle Fusion Middleware

ORACLE

Executive Overview

The ability to understand and control the nature of software within the enterprise is critical for today's large organizations. Understanding what users are running on their desktops enables IT departments to manage security and liability risk as well as facilitating the implementation of IT policy around software updates and upgrades.

The Java runtime environment is the most widely used application runtime platform deployed on enterprise systems today. In addition to being installed on more than 97% of enterprise desktops, the Java virtual machine powers a large percentage of server-based packaged and custom applications as well as a diverse set of mobile and embedded platforms.

This paper describes an approach for enabling the tracking and visualization of enterprise Java usage through the use of Oracle Fusion Middleware. Organizations adopting such an approach can be expected to realize a diverse set of benefits including:

- Insight into Java installations and deployments across locations
- Ability to avoid liability issues and stay on top of security updates with an efficient tracking system
- Lower internal support costs for tracking and maintaining multiple versions of Java
- Increased manageability of systems and applications with a visual, real-time dashboard of Java deployments
- Enablement of proactive IT operations with reports, alerts, and the ability to perform usage data forensics

Introduction

The challenge of creating a distributed, reliable, real-time, and enterprise grade visualization and analysis application for the tracking of Java runtime environment (JRE) usage can be easily accomplished through the use of tools within Oracle's Fusion Middleware toolbox.

The benefit of this approach is that it allows organizations to build the infrastructure for such a solution using enterprise-grade components with minimal software development while maintaining a flexible visualization and analytics layer that can easily be customized for each organization's unique business needs in a declarative fashion. It is possible to build a similar solution through custom coding or scripting by mixing a variety of custom or off-the-shelf components; however doing so would most certainly take more time, require more coding and configuration, be less resilient to software failures, and not provide the same level of configurability and changeability by the non-technical user.

In addition to serving as a vehicle for providing insight, the tools in the Fusion Middleware toolbox can also be leveraged to integrate actionable hooks into the analytics environment. For example, an administrator seeing a pattern of questionable program invocations or out-of-compliance JRE instances can initiate an internal service request to investigate and rectify the usage directly from the dashboard.

The reference architecture proposed in this paper utilizes the following components of Fusion Middleware:

- Oracle Java SE Advanced
- WebLogic Enterprise Edition
- Business Activity Monitoring (BAM)
- Business Process Management (BPM) Suite

It is intended that organizations use the concepts and approach described in this paper as a template and inspiration for an implementation that can meet their own unique needs.

The Building Blocks

This solution can be constructed exclusively through the use of Oracle Fusion Middleware components. Oracle Fusion Middleware is a collection of standards-based software products that spans a range of tools and services: from Java EE and developer tools, to integration services, identity management, business process management, business intelligence, and collaboration. Oracle Fusion Middleware offers complete support for development, deployment, and management.

The following describes the palette of tools used to build a comprehensive solution. Depending on the requirements of the individual organization, a subset of these tools may be necessary to achieve that organization's business and technical needs; for example, organizations interested only in insight and visualization but not in driving integrated action on the part of their users may not need to consider the BPM Suite.

Oracle Java SE Advanced

Oracle Java SE Advanced offers a set of commercial features above the standard functionality of the Java Runtime Environment (JRE) and Java Development Kit (JDK) which includes tools to allow enterprise organizations to gain greater visibility and control over large installations by knowing exactly which versions of Java are deployed and running what applications. In addition, Oracle Java SE Advanced provides a number of other key features that enable low-impact virtual machine monitoring on production systems and after-the-fact incident logging and debugging.

One of these commercial features: Usage Tracking, allows the Java Runtime Environment (JRE) to be configured to collect telemetry data for each Java SE invocation on every desktop or server in the enterprise. The data collected includes the following:

- Java and JVM versions
- Java application name
- Host name and IP address
- Invocation timestamp
- JVM arguments
- Classpath
- JRE location on the system
- Operating system name and version
- Launch type (e.g. applet, web start, command line)

By providing a configuration file to the JRE, this information can be captured and recorded on every invocation and can either be stored locally on host or delivered over the network to a central tracking server.

More details about Oracle Java SE Advanced can be found at:
http://www.oracle.com/us/technologies/java/standard-edition/advanced-suite/overview/index.html

WebLogic Enterprise Edition

Oracle WebLogic Server Enterprise Edition is a scalable, enterprise-ready Java Platform, Enterprise Edition (Java EE) application server and forms the base infrastructure layer for Oracle Fusion Middleware. WebLogic Server enables enterprises to deploy mission-critical applications in a robust, secure, highly available, and scalable environment and facilitates the configuration of clusters to distribute load, and provide extra capacity in case of hardware or other failures.

Within the context of this solution, WebLogic serves as the application container that hosts the higher level applications as well as the custom Java code required to consume Java usage tracking data. WebLogic provides clustered singleton support which facilitates the reliable nature of this solution as well as Java Message Service (JMS) support which allows for the asynchronous yet reliable routing of usage tracking data to the visualization engine.

More details about Weblogic Server can be found at:
http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html

Business Activity Monitoring (BAM)

Oracle Business Activity Monitoring (BAM) is a complete solution for building interactive, real-time dashboards and proactive alerts for monitoring business processes and services. BAM gives both business executives and operational managers timely information to make better decisions. Real-time event updates allow users to gauge - within seconds - the impact of key performance indicators on their business and take immediate corrective actions

BAM provides non-developers with the capability to create custom dashboards without having to write a single line of code. Through a step-by-step wizard, business users can create dashboards that monitor and correlate real-time events.

Within the context of this solution, BAM serves as the end-user facing data acquisition and visualization engine.

More details about BAM can be found at:
http://www.oracle.com/technetwork/middleware/bam/overview/index.html

Business Process Management Suite (BPMS)

Oracle Business Process Management Suite (BPMS) is a complete product suite that leverages industry standard languages and notations for execution of processes by a unified engine. It provides a complete process lifecycle with modeling, managing, simulating, optimizing, and executing business processes across organizational divisions, systems, and applications.

Within the context of this solution, BPMS can be leveraged to automate the response to insights surfaced by BAM regardless of whether that response involves human or system tasks.

More details about BPMS can be found at:
http://www.oracle.com/technetwork/middleware/bpm/overview/index.html

The Architecture – Putting the Building Blocks Together

The Fusion Middleware building blocks described in the previous section can be assembled, along with some configuration and a bit of custom code, into a solution architecture that achieves the goal of tracking JRE usage in a distributed, reliable, real-time, and actionable fashion.

Figure 1 visually depicts the Java usage tracking solution architecture. The context diagram is numerically annotated with the steps necessary to realize the solution which are described in greater detail further in this section.

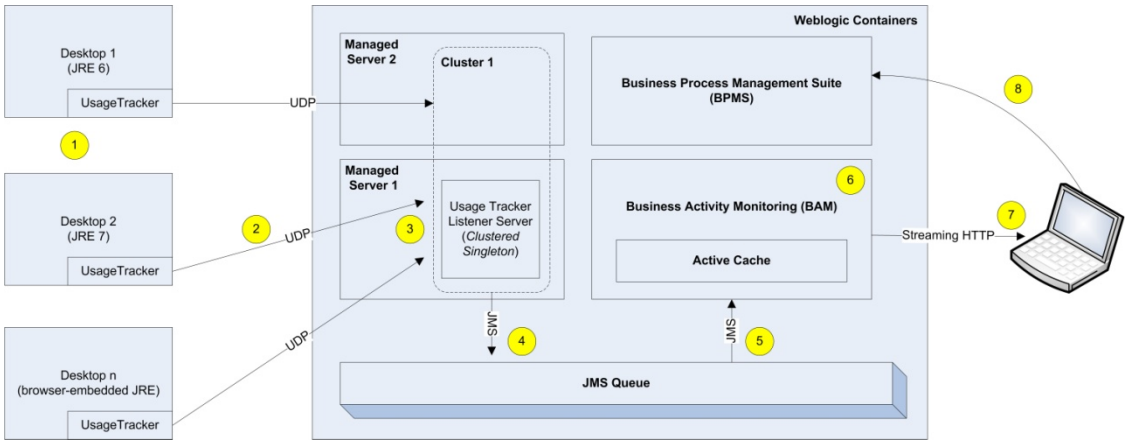


Figure 1: Solution Architecture

Step 1: Java Runtime Configuration

Each licensed Java Runtime Environment (JRE) must be enabled for usage tracking. This is achieved by adding a *usagetracker.properties* file to the *$JAVA_HOME/lib/management* directory where *$JAVA_HOME* represents the JRE home directory. The *usagetracker.properties* file should contain a

single line that reads *com.oracle.usagetracker.logToUDP = $host:$port* where *$host* represents the DNS name or IP address of the centralized usage tracking server (described in subsequent sections) and *$port* represents the UDP port on which that centralized server accepts connections; the default listen port is 32139 but the implementer may select any listen port as long as it is consistent across all 'client' JREs and the centralized server.

In enterprise environments, this property file can be distributed en-masse by rolling out a pre-configured snapshot of the JRE or by automatically updating existing environments through the use of systems management tools like Microsoft's System Center Configuration Manager (SCCM).

This configuration can be applied to any JRE whether it is present on an end user's desktop or on a back office server.

Further details about usage tracker configuration can be found at:
http://docs.oracle.com/javase/products/usagetracker.html

Step 2: Usage Tracking Initiation

Each time a configured JRE is invoked the usage tracker feature serializes information about the invocation as a UDP packet and sends it to the server/port combination specified in the *usagetracker.properties* file.

The type of data tracked for each invocation includes Java version/vendor, JVM version/vendor, application name, host name, classpath, JVM arguments, and timestamp. Full details about what information is logged can be found at: http://docs.oracle.com/javase/products/usagetracker.html.

Step 3: Usage Tracking Server

The role of the usage tracking server in this architecture is to capture invocation datagrams enterprise-wide and convert them into a form that can be processed by the data visualization engine. This involves deploying a central server which is capable of capturing and decoding these packets. A simplified sample implementation of such a server can be found at:
http://docs.oracle.com/javase/products/samplecode/UsageTrackerServer.java.

While such a server could be implemented in a standalone fashion, this approach would require manual activation, monitoring, and would not be fault tolerant in the event of software or hardware failure. To solve for these issues, this architecture implements the usage tracker server as a Clustered Singleton service hosted by WebLogic Enterprise Edition. This approach allows for the usage tracker to be hosted on a single server within a WebLogic cluster, activated automatically upon WebLogic server startup, and automatically migrated to another hardware node in the cluster in case of software or hardware failure.

A clustered singleton can be developed by implementing the *weblogic.cluster.singleton.SingletonService* interface and bootstrapping the usage tracker server's *run()* method from the singleton's *activate()*

method. The singleton along with supporting classes should be packaged as a JAR and copied into the WebLogic domain's *lib* folder on all application servers within the cluster. The cluster should then be configured to recognize the singleton via WebLogic Scripting Tool (WLST) scripts or via the admin console as shown in Figure 2.

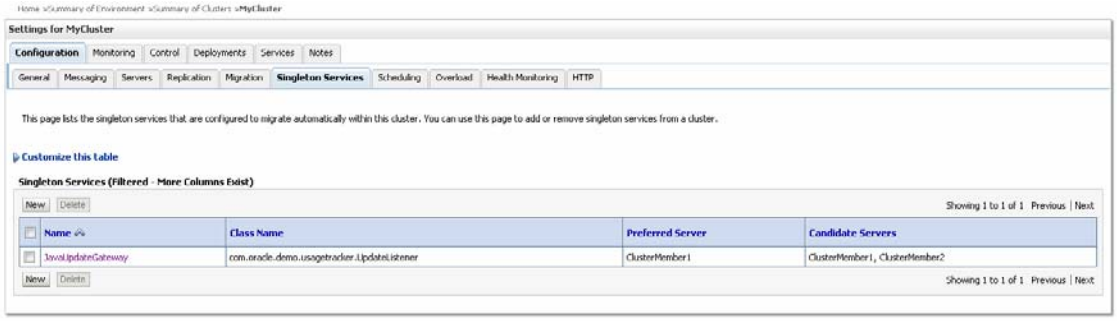


Figure 2: Clustered Singleton Configuration through WebLogic Console

For a clustered singleton to be activated at server startup and migrated upon server failure, WebLogic managed server instances that are participating in the cluster hosting the singleton **must** be started by WebLogic's Node Manager. Details on configuring and operating Node Manager can be found here: http://docs.oracle.com/cd/E23943_01/web.1111/e13740/toc.htm.

Step 4: Transform and Integrate

At this stage, usage data has been received by the usage tracking server via UDP. However, to make it available to the visualization engine, that data must be converted to XML and placed onto a Java Message System (JMS) queue. Use of JMS serves as a reliable and loosely-coupled integration pattern between the custom usage tracking server and the packaged visualization engine implemented with Oracle Business Activity Monitoring (BAM).

WebLogic Enterprise Edition includes an enterprise-grade JMS infrastructure which should be used to create a new JMS queue. The usage tracking server code is responsible for parsing the incoming UDP messages as they arrive by transforming them to a simple XML structure, and using WebLogic's JMS application programming interface (API) to inject them into the newly created JMS queue. Because the UDP messages arrive in comma-separated (CSV) form in a well known order (as detailed in the documentation referenced in step 2) parsing them simply involves splitting the string on commas and building an XML string that looks like Figure 3:

```
<?xml version="1.0" encoding="UTF-8"?>
<UsageLine>
  <Type>VM start</Type>
  <DateTime>Mon Apr 01 15:42:16 EST 2013</DateTime>
  <Hostname>ESHNEKEN-US/12.345.678.987</Hostname>
  <AppName>com.sun.deploy.panel.ControlPanel</AppName>
  <JrePath>C:\Program Files\Java\jre7</JrePath>
  <JavaVersion>1.7.0_11</JavaVersion>
  <JvmVersion>23.6-b04</JvmVersion>
  <JavaVendor>Oracle Corporation</JavaVendor>
  <JvmVendor>Oracle Corporation</JvmVendor>
  <OsName>Windows 7</OsName>
  <OsArch>amd64</OsArch>
  <OsVersion>6.1</OsVersion>
  <JvmArgs>-Xbootclasspath/a:C:\Program Files\Java\jre7\bin\..\lib\deploy.jar -Duser.home=C:\Users\eshneken </JvmArgs>
  <ClassPath>.;C:\Java\jre6\lib\ext\QTJava.zip</ClassPath>
</UsageLine>
```

Figure 3: Usage Tracker message after conversion to XML

More details about configuring JMS under WebLogic can be found at: http://docs.oracle.com/cd/E23943_01/web.1111/e13738/toc.htm.

Step 5: Load Data to Active Cache

Oracle Business Activity Monitoring (BAM) is a complete solution for building interactive, real-time dashboards and proactive alerts for monitoring business processes and services. The first step to leverage the power of BAM to serve as a visualization engine for this solution is to define a data structure that matches the data provided by the Oracle Java SE Advanced Usage Tracking implementation. Figure 4 shows a data object layout defined using BAM's *Architect Administration Console* that supports usage tracking.

Field name	Field ID	Field type	Max length	Scale	Nullable	Public	Lookup	Calculated	Tip Text
Type	_Type	string	50	-	Yes	Yes	-	-	Invocation Type
DateTime	_DateTime	datetime	-	-	Yes	Yes	-	-	Date/Time
AppName	_AppName	string	500	-	Yes	Yes	-	-	Class Name
JrePath	_JrePath	string	500	-	Yes	Yes	-	-	JRE Home Directory
JavaVersion	_JavaVersion	string	50	-	Yes	Yes	-	-	Java Version
JvmVersion	_JvmVersion	string	50	-	Yes	Yes	-	-	JVM Version
JavaVendor	_JavaVendor	string	100	-	Yes	Yes	-	-	Java Vendor
JvmVendor	_JvmVendor	string	100	-	Yes	Yes	-	-	JVM Vendor
OsName	_OsName	string	50	-	Yes	Yes	-	-	Operating System Version
OsArch	_OsArch	string	50	-	Yes	Yes	-	-	Operating System Architecture
OsVersion	_OsVersion	string	100	-	Yes	Yes	-	-	Operating System Version
JvmArgs	_JvmArgs	string	100	-	Yes	Yes	-	-	JVM Arguments
ClassPath	_ClassPath	string	200	-	Yes	Yes	-	-	ClassPath
Hostname	_Hostname	string	100	-	Yes	Yes	-	-	Hostname/IP

Figure 4: BAM data object configuration

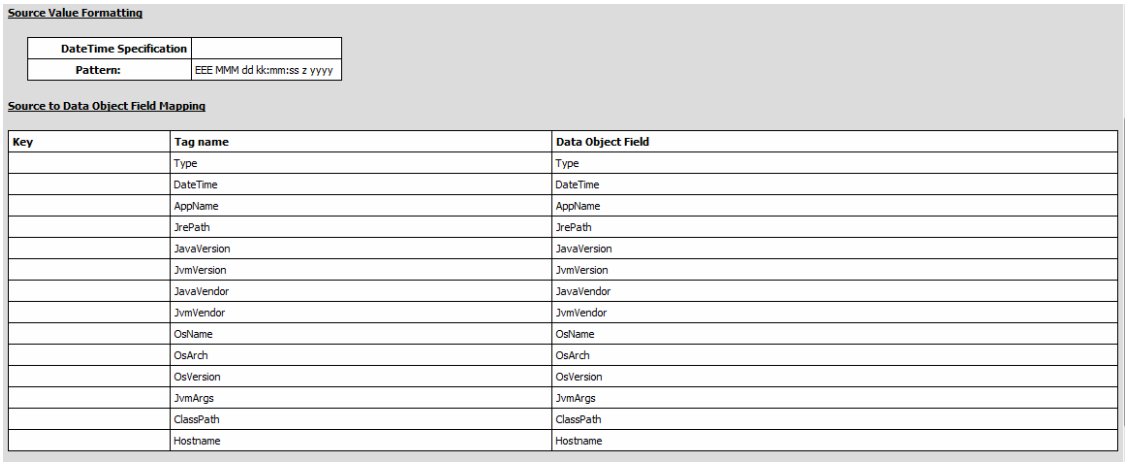
Data objects that are defined in BAM are stored within BAM's *Active Data Cache* which is a persistent, transacted, memory-based storage system that makes data available for access by BAM's *Active Report Engine* for real-time delivery to end users. There are a number of ways that data can be injected into the *Active Data Cache*; for the purpose of this architecture, the ingress method is linkage of a JMS queue directly to the data object configured in the Active Data Cache.

By using the *BAM Architect Console* an *Enterprise Message Source* can be configured that links a JMS queue to a data object and defines any transformation that may be needed. Figure 5 shows the definition of such a message source.

Initial Context Factory:	weblogic.jndi.WLInitialContextFactory
JNDI Service Provider URL:	t3://localhost:7001
Topic/Queue Connection Factory Name:	jms/UsageTrackerQueueConnectionFactory
Topic/Queue Name:	jms/usageTrackerQueue
JNDI Username:	
JNDI Password:	
JMS Message Type:	TextMessage
Durable Subscriber Name (Optional):	
Message Selector (Optional):	
Data Object Name:	/UsageTracker/InvocationInstance
Operation:	Insert
Batching:	No
Transaction:	No
Start when BAM Server starts:	Yes
JMS Username (Optional):	
JMS Password (Optional):	

Figure 5: Enterprise Message Source tying a local JMS queue to a BAM data object

Figure 6 shows the mapping between XML element names in the JMS message and the field names defined in the data object. In addition, some data types (like *DateTime* objects) may require special handling and this figure also shows how an appropriate *DateTime* format can be specified.



Source Value Formatting

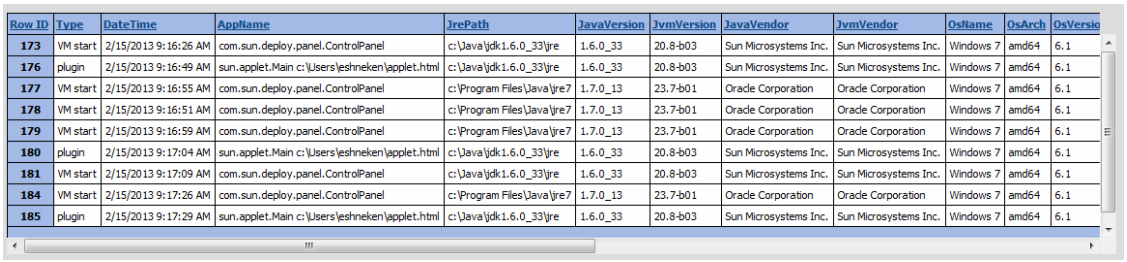
DateTime Specification	
Pattern:	EEE MMM dd kk:mm:ss z yyyy

Source to Data Object Field Mapping

Key	Tag name	Data Object Field
	Type	Type
	DateTime	DateTime
	AppName	AppName
	JrePath	JrePath
	JavaVersion	JavaVersion
	JvmVersion	JvmVersion
	JavaVendor	JavaVendor
	JvmVendor	JvmVendor
	OsName	OsName
	OsArch	OsArch
	OsVersion	OsVersion
	JvmArgs	JvmArgs
	ClassPath	ClassPath
	Hostname	Hostname

Figure 6: Enterprise Message Source configuration showing message to data object field mapping and transformation

Once the data object and message sources are configured and the message source is activated, all Java invocations across the enterprise that have been properly configured will pass their data to the Java usage tracking server via UDP which will transform that data to XML and pass the data to BAM via a JMS queue. The raw form of that data can now be centrally viewed in the Active Data Cache by looking at the data object's contents using BAM's *Architect Console* as seen in Figure 7.



Row ID	Type	DateTime	AppName	JrePath	JavaVersion	JvmVersion	JavaVendor	JvmVendor	OsName	OsArch	OsVersio
173	VM start	2/15/2013 9:16:26 AM	com.sun.deploy.panel.ControlPanel	c:\Java\jdk1.6.0_33\jre	1.6.0_33	20.8-b03	Sun Microsystems Inc.	Sun Microsystems Inc.	Windows 7	amd64	6.1
176	plugin	2/15/2013 9:16:49 AM	sun.applet.Main c:\Users\eshneken\applet.html	c:\Java\jdk1.6.0_33\jre	1.6.0_33	20.8-b03	Sun Microsystems Inc.	Sun Microsystems Inc.	Windows 7	amd64	6.1
177	VM start	2/15/2013 9:16:55 AM	com.sun.deploy.panel.ControlPanel	c:\Program Files\Java\jre7	1.7.0_13	23.7-b01	Oracle Corporation	Oracle Corporation	Windows 7	amd64	6.1
178	VM start	2/15/2013 9:16:51 AM	com.sun.deploy.panel.ControlPanel	c:\Program Files\Java\jre7	1.7.0_13	23.7-b01	Oracle Corporation	Oracle Corporation	Windows 7	amd64	6.1
179	VM start	2/15/2013 9:16:59 AM	com.sun.deploy.panel.ControlPanel	c:\Program Files\Java\jre7	1.7.0_13	23.7-b01	Oracle Corporation	Oracle Corporation	Windows 7	amd64	6.1
180	plugin	2/15/2013 9:17:04 AM	sun.applet.Main c:\Users\eshneken\applet.html	c:\Java\jdk1.6.0_33\jre	1.6.0_33	20.8-b03	Sun Microsystems Inc.	Sun Microsystems Inc.	Windows 7	amd64	6.1
181	VM start	2/15/2013 9:17:09 AM	com.sun.deploy.panel.ControlPanel	c:\Java\jdk1.6.0_33\jre	1.6.0_33	20.8-b03	Sun Microsystems Inc.	Sun Microsystems Inc.	Windows 7	amd64	6.1
184	VM start	2/15/2013 9:17:26 AM	com.sun.deploy.panel.ControlPanel	c:\Program Files\Java\jre7	1.7.0_13	23.7-b01	Oracle Corporation	Oracle Corporation	Windows 7	amd64	6.1
185	plugin	2/15/2013 9:17:29 AM	sun.applet.Main c:\Users\eshneken\applet.html	c:\Java\jdk1.6.0_33\jre	1.6.0_33	20.8-b03	Sun Microsystems Inc.	Sun Microsystems Inc.	Windows 7	amd64	6.1

Figure 7: A view of usage tracking data object content

Step 6: Declaratively Construct Custom Dashboards

Oracle Business Activity Monitoring (BAM) can be used by business users to visually and declaratively build and deploy real-time monitoring dashboards. BAM's *Active Studio Console* can be leveraged to construct a multi-layer graphical dashboard tied to the data object and message source configured in the previous step.

Dashboard creators can choose from a rich set of graph and table types to support display of a variety of metrics related to Java usage tracking. Some samples uses include:

- Tracking total JVM invocations by version number
- Breakdown of JVM invocations by type (applet, application, web start, etc)

- Number of hosts in the enterprise segmented by version
- Most utilized Java applications

All elements of the raw data object are available for utilization and the ability exists to augment the core dataset either through creation of 'calculated fields' that are based on some logical transformation of the core data or through the combination of the core data with other enterprise data sources specific to the implementing organization that can be integrated into BAM. In this example, we have created a 'calculated field' that determines whether a JVM version is in-compliance with corporate IT policy and display that metric on the dashboard.

A key business benefit of this approach is that sophisticated and visually compelling dashboards can be built and modified as requirements change very quickly through a web interface by non-programmers without the need for engaging graphic designers.

More details on creating and deploying BAM reports can be found at: http://docs.oracle.com/cd/E23943_01/user.1111/e10230/toc.htm.

Step 7: Deliver Real-Time Dashboards to the End User

Once the dashboard has been created, it is available to users to view in their web browser directly or as a region (or portlet) within an internally or externally facing portal. Oracle BAM streams data directly to the dashboard so users can keep the page with the dashboard open and it will update in real-time as new data arrives in the *Active Data Cache* without the need for the user to refresh the page. Figure 8 shows the end user's view of a sample Java usage tracking dashboard.

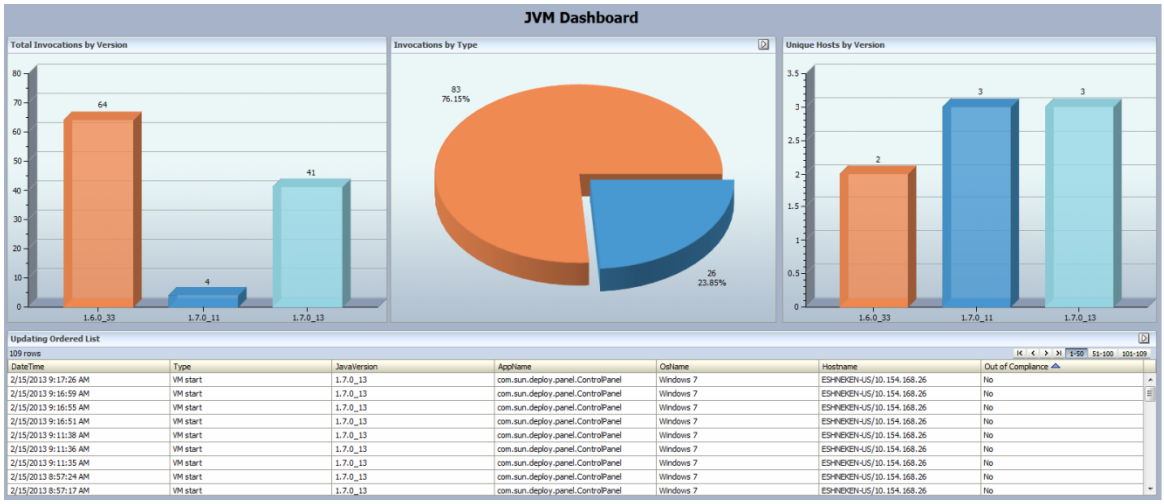


Figure 8: End user's view of the Java usage tracking dashboard

Step 8: Turn Insight into Action

The solution up to this point is certainly compelling enough; enterprise IT executives and operational managers now have the power to monitor usage of their company's Java footprint centrally, reliably, and in a real-time fashion. In addition, this solution is nimble enough to be tweaked by non-technical users as requirements and monitoring patterns evolve.

However, the real power of such a solution is the ability to closely tie insight to action whereby users who detect a troubling pattern or condition through the dashboard are able to immediately take steps to rectify the situation by kicking off a mitigating process directly from inside the dashboard.

BAM has the ability to initiate external HTTP calls from user triggered buttons on the dashboard. These calls can be used to start a business process hosted on Oracle's Business Process Management Suite (BPMS). The business process can contain diverse elements like business rules, system-level integrations, and human-based approvals that exhibit the characteristics of being automated and traceable. For example, an end user noticing an out-of-compliance JRE or a deprecated application being run on a user's machine can immediately click a button next to the entry in the BAM dashboard and trigger a business process which opens a case with IT and automatically routes to an administrator. Upon reviewing the usage, the administrator can choose to automatically trigger an update to the offending user's desktop by triggering an interaction with the company's systems management application.

Insight on its own is powerful, but insight paired with tightly integrated actionable hooks translates to even higher levels of business process success.

Conclusion

IT organizations looking to reduce risk, lower support costs, and gain actionable insight around their Java SE deployments can do so in a distributed, reliable, and real-time fashion by leveraging various components of the Oracle Fusion Middleware stack. The advantage of following this blueprint and architecting the solution using these components is that this solution is flexible enough to be easily customized to the specific needs of each organization while at the same time being easy to develop and deploy with a minimal amount of custom coding.

Java Usage Tracking and Visualization with Oracle Fusion Middleware
May 2013
Author: Edward Shnekendorf

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com