

WebSphere for z/OS

Application Debugging and Profiling

It is possible and actually quite easy to use remote debugging and profiling on J2EE applications deployed in a WebSphere for z/OS and OS/390 Version 4.0.1 server. The intent of this paper is to show how you can accomplish both of these tasks using a modified version of the Policy IVP application which ships with the WebSphere for z/OS Version 4.0.1 (WAS for z/OS) product. This discussion assumes that you, the reader, have knowledge of:

1. The WAS for z/OS runtime, in particular you know how to setup a server similar to the one initially used to install the Policy IVP and install know how to install an application in a server.
2. The WebSphere Application Developer (WSAD) tool set.
3. The Jinsight for Java2 profiling and visualization tool set (Jinsight 2.1) as distributed from <http://www.alphaworks.ibm.com>.

If this is not the case, then you have a substantial amount of reading to do before starting on this expedition. This paper is not a tutorial on all of these tools, merely an example of how you can use these tools in the WAS for z/OS environment.

From the author's perspective, the reasons for this paper are:

1. Convert the Policy IVP application as distributed by WAS for z/OS from a Visual Age for Java (VAJ) project to a WebSphere Application Developer (WSAD) project.
2. Use the WSAD provided wizards to create a WebService over the PolicySession.trans1() method and deploy this web service as part of the new Policy IVP application in a WAS for z/OS server.
3. Demonstrate the setup and use of the WSAD product's capability to do step-by-step application debugging and application profiling using the new version of the Policy IVP in a WAS for z/OS server.
4. Incorporate the Jinsight controller web application into a J2EE application, and demonstrate the setup and use of the Jinsight tool's capability to do application profiling using the new version of the Policy IVP in a WAS for z/OS server.

In as much as this an early attempt to use WSAD for a J2EE project, a number of decisions were made which may not have been optimal choices in retrospect.

Policy IVP Notes:

1. Each of the EJBs resides in a separate module or .jar file rather than having all the EJBs reside in a single .jar file.
2. The Policy Utility classes are in a java project separate from any of the web application .war files or the EJB .jar files associated with the Policy IVP J2EE application.
3. The JNDI reference names of the components were changed slightly from the original names. For example the Was40Ivp servlet will lookup "java:comp/env/ejb/policysession" instead of "java:comp/env/ejb/ivp.policysession". There is nothing significant to this change, it just happened during the process.

WebSphere for z/OS
Application Debugging and Profiling

4. /PolicyIVP2 is the context root for this new version of the Policy IVP web application. This decision was made so that the original Policy IVP application could coexist with the new version in the situation where the IBM HTTP Server and the WAS 4.0.1 plug-in are used to invoke web applications. This decision required changes to the cebit.html, cebit.jsp and Was40Ivp.java files, since the context root was hard coded into the application.
5. The CMP bean (i.e., PolicyCMP) has been mapped to a table using a schema (i.e., BBO) just like the BMP bean. Thus the problem of having to assign an alias to the underlying DB2 table (i.e., BBO.POLICYDO) so the server can access the table is avoided.
6. The Web Service Wizard generates an .xmi document that contains the JNDI name of the PolicySession bean as deployed in the WSAD. To make it easier to deploy this application in your own server, the dds.xml file, which is generated by the WebService Wizard, was modified to set: `<isd:option key="JNDIName" value="java:comp/env/ejb/ivp/policysession" />`.
7. None of the EJB methods throwing the java.remote.Exception were fixed; WSAD will issue warnings for this oversight.

Jinsight Notes:

1. The Jinsight web application was modified so that it could be invoked from any URI. This required a small change to the jintrace.htm file. Only the web application is distributed as part of this paper in an attempt to reduce the size of the file.
2. The Jinsight control web application was assigned a context root of /PolicyIVP2x. This was done because one cannot use the same context root for web applications residing in different .war files even if deployed in the same server.

Miscellaneous Packaging Notes:

1. The Jinsight controller web application is packaged as a separate J2EE application and not as part of the Policy IVP web application. This will allow you to easily uninstall the Jinsight application from the server, once you are convinced the behavior of the application is as expected, without having to uninstall the Policy IVP2 application or change the packaging. You can easily install it any other server by changing the context root appropriately using AAT for z/OS and installing the .ear file in the target server with the default JNDI name for the web application.
2. The PolicyIVP2 and Jinsight applications were exported from WSAD as J2EE applications (i.e., ear files). However, both of these .ear files must be processed by the AAT for z/OS tool prior to installation into the WAS for z/OS runtime.
3. AAT for z/OS required altering the context root on web apps, they needed to be pre-pended with a forward slash (i.e., '/). Additionally, the EJB references and resource references were updated to use a link, making deployment easier.
4. As usual, the vaprt.jar, ivjejb35.jar and PolicyUtilities.jar files were manually included in the .ear file emitted from WAS for z/OS AAT tool as they were not included in the .ear file emitted by WSAD.

Prerequisites

To use these debugging and profiling tools on the new Policy IVP application, one must first install some small amount of code on the workstation and on z/OS.

Workstation:

You need a WINNT, WIN2K or WINXP workstation with the following installed:

1. IBM Java SDK 1.3

You can obtain this package from <http://www.ibm.com/developerworks/java>, follow the links from the label: IBM developer kits.

2. WebSphere Application Developer 4.0.2
3. Jinsight 2.1

You can obtain the Jinsight2.1 package from the website: <http://www.alphaworks.ibm.com>. Point your favorite web browser at this URL, search for "jinsight", go to the download page and retrieve the jinsight2.1-os390-java2.zip package. There is no cost to acquire/use this package.

Then unzip the file into directory a directory of your choice (i.e., d:\Program Files\IBM\), which will create a directory named jinsight2.1 and subsequent directories. Be certain to read the documentation in the .doc directory. This package has the visualization tool set, which will run on the workstation and the code needed on z/OS to collect the application profiling data.

4. Assembling Application Tool for z/OS

You will need Driver 23 or higher to create the .ear file should you decide to do so. The AAT for z/OS tool must be acquired from the WebSphere website (i.e., http://www.ibm.com/software/webservers/appserv/download_v4z.html). Retrieve the installable package and install the product in the directory of your choice.

5. WebSphere for z/OS Systems Management Administration Tool

Install the level of bbonisnst.exe that corresponds to your level of WAS for z/OS. It is located in the <WebSphere_for_zOS_Install_Root>/bin directory on the z/OS system where WAS for z/OS is installed. FTP this package in BINARY mode to your workstation and install in the directory of your choice.

6. (Optional) UDB 7.1 with PTF 3 or higher

To use the WSAD test environment you need UDB. However to use the distributed debugger or profiling tools it is not required.

Included along with this document is the PolicyIVP V2.zip file. This .zip file contains all the parts needed to install, debug and profile the updated Policy IVP with the changes outlined earlier.

Retrieve the file and unzip the PolicyIVP V2.zip file into a directory of your choice (i.e., D:\zzz\IVP401), being certain to keep the folder names. You will discover the following contents:

WebSphere for z/OS
Application Debugging and Profiling

1. The WSAD workspace directory, appropriately named: workspace. This workspace includes the revised version of the Policy IVP and the Jinsight control web application.
2. A directory containing all the parts to construct the application .ear file to be deployed in WAS for z/OS, it is named: WSAD_J2EEParts. The parts contained within can be used to build an .ear file using AAT for z/OS if you should desired to do so.
3. The original .ear files exported from WSAD are also located in this directory with the name: PolicyIVP2.ear and PolicyIVP2-jinsight.ear. These .ear files cannot be deployed into the WAS for z/OS 4.0.1 runtime, but can be used as input into the AAT for z/OS tool.
4. The J2EE application .ear files containing the Policy IVP and Jinsight applications that are ready to be deployed into the WAS for z/OS server:
 - WSAD_J2EEParts\ PolicyIVP2_zOS.ear, containing the new Policy IVP J2EE application.
 - WSAD_J2EEParts\ PolicyIVP2-jinsight_zOS.ear, containing the Jinsight web application.
5. A .bat file to invoke WSAD application using the included workspace: WSAD-IVP.bat

You can start the WSAD tool with the provided workspace and export all the J2EE application parts and construct deployable .ear files using the Assembling Application Tool for z/OS (AAT) at driver level 23 . This is left as a "user exercise". If you choose to use the pre-built parts, you can and should examine all of the parts in WSAD and AAT.

Recommendation: For the first attempt in using the WSAD distributed debugger profiling tools and Jinsight profiling tools, you should install the pre-built .ear file containing the new version of the Policy IVP J2EE application in the WAS for z/OS server of your choice.

Host:

You will need a WAS for z/OS application server into which you can install this revised application. You can use the same server where the original Policy IVP is installed, merely create a new conversation and install the new version of the Policy IVP application and the Jinsight application using default JNDI names. Since the packaging of the application is slightly different than the package distributed with the product, the default JNDI names will be sufficiently different and the applications will actually coexist in the same server.

However, the recommendation is to create a new server, just to be safe. For this exercise, a new server named WSPMR1 was created to contain this version of the Policy IVP application. For the WSPMR1 server:

- ◆ the associated control region is named: WSPMR1C with userid: WSPMR1C,
- ◆ the server region is named: WSPMR1S with userid: WSPMR1S, and
- ◆ default local and remote identity: WSGUEST, and
- ◆ the server instance is named: WSPMR1A1.

This new server was defined much like the original server into which the original Policy IVP was installed. If you choose to create a new server, you must remember to allow the userid associated with the server region (i.e., WSPMR1S) access to the BBO.POLICYDO table, just as you did the server that housed the original Policy IVP J2EE application, only there is no need for an alias on the table. Of course all the normal security definitions, JCL PROCs, etc, must be in place for this new server.

/PolicyIVP2x as the context root for the Jinsight controller web application and /PolicyIVP2 as the context root for the PolicyIVP web application. If you are using the IBM HTTP Server and the WAS 4.0.1 plugin to route requests to this WAS for z/OS server (i.e., WSPMR1) you must update the service statements in the httpd.conf file to add service directives for: /PolicyIVP2/* and /PolicyIVP2x/*. For this reason, it is

WebSphere for z/OS Application Debugging and Profiling

recommended that you setup the new server to use the HTTP Transport Protocol capability in the server control region to terminate the http session. After you acquire a port for your server instance from the guardians of TCPIP (i.e., 8887), you should set the following HTTP related variables in the environment variables for the WSPMR1A1 server instance:

```
BBOC_HTTP_SESSION_GC=0
BBOC_HTTP_INPUT_TIMEOUT=0
BBOC_HTTP_OUTPUT_TIMEOUT=0
BBOC_HTTP_PORT=8887
```

If you are only setting up a single server instance, you can see these values at the server level. These variables will be reflected in the current.env file for the server instance (i.e., /WebSphere/WAS401/controlinfo/envfile/WSCPLEX/WSPMR1A1/current.env).

When defining the server for this debugging/profiling exercise:

1. You must specify the debugger is allowed when defining your server.
2. You should allow multiple transactions to run concurrently in a server AND allow only one server per control region.

The server instance requires setting up the normal webcontainer.conf, jvm.properties, and trace.settings files appropriately as with any other WAS for z/OS J2EE server. In particular set the virtual host name correctly in the webcontainer.conf file. The contents of these files are not shown, this is just a reminder to do this task before continuing.

As a general rule, the recommendation is to create a new conversation defining an "empty server" prior to installing any applications in the server. This can be a fairly well canned process and you are certain the server ready and functioning prior to installing applications into it.

Application Installation:

The first task is to install the two J2EE applications into the server of your choice (i.e., WSPMR1). You must install both the PolicyIVP2-jinsight_zOS.ear and PolicyIVP_zOS.ear files from the WSAD_J2EEParts directory. Install both J2EE applications in a single conversation.

All that must be done when installing the J2EE applications into the server is:

1. Take the default JNDI names for all the EJBs and Web Apps.
2. Select the same DB2 data source for the PolicyBMP and PolicyCMP beans in this server as was used in the original Policy IVP installation in the J2EE Resource Reference pages for these two EJBs.

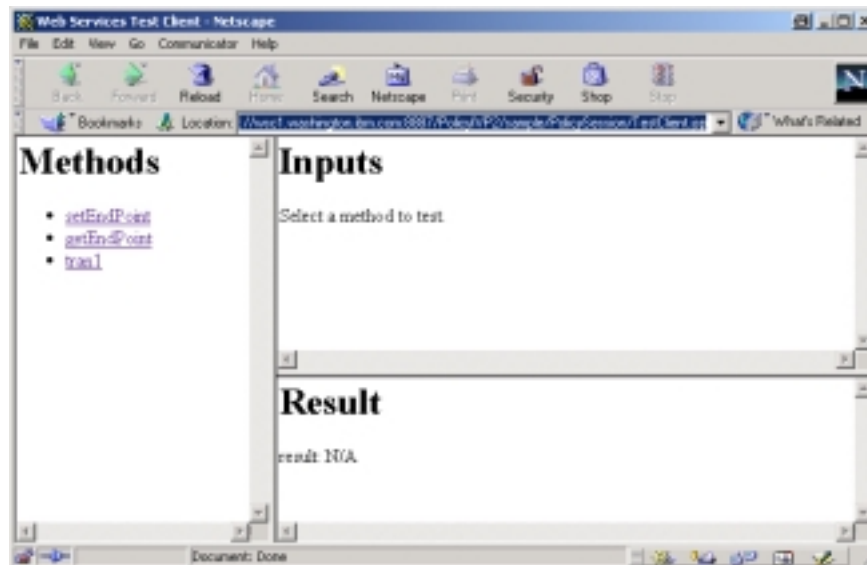
No other special handling of the .ear files is required.

Once the conversation is activated and the J2EE applications are installed and registered, use the browser of your choice invoke the URL for the Policy IVP web application (i.e., <http://wsc1.washington.ibm.com:8887/PolicyIVP2/cebit.html>) and verify the modified Policy IVP application does indeed work prior to proceeding.

At this time you can verify the Jinsight main page can be loaded/displayed by invoking the URL: <http://wsc1.washington.ibm.com/PolicyIVP2x/jintrace.htm>. The application cannot be run as it is not fully configured, but this should indicate that the WAS for z/OS server can find the main page.

Also, at this time you might try to use the web service by invoking the web application that builds the SOAP message to be passed to the rpcrouter servlet (i.e., <http://wsc1.washington.ibm.com:8887/PolicyIVP2/sample/PolicySession/TestClient.jsp>).

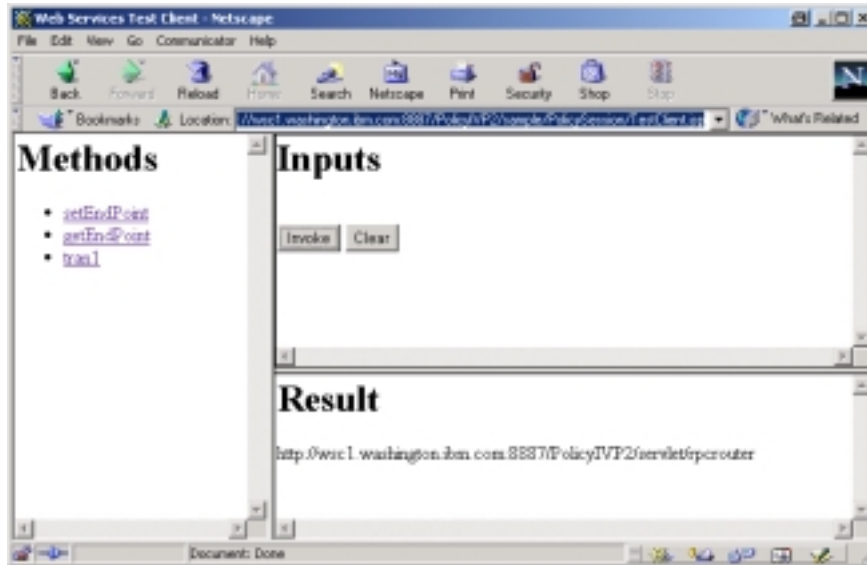
You should see the following:



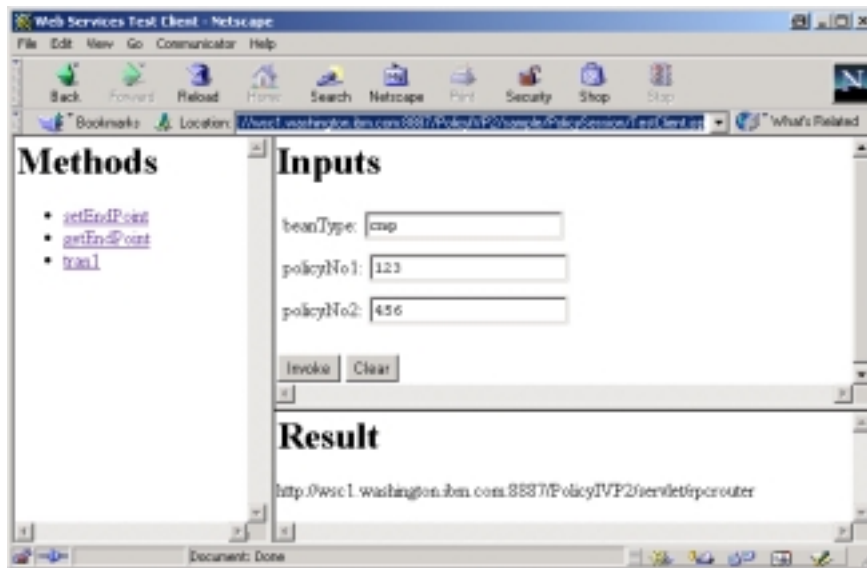
The servlet with which you are interacting (the sending servlet) will create a soap message and send it in an HTTP request to the rpcrouter servlet that will in turn invoke the tran1() method on the PolicySession EJB session bean. However, the default location (or URL) for the rpcrouter servlet in the sending servlet is not correct. In order to send the SOAP message to the rpcrouter servlet in your server you must set the correct

WebSphere for z/OS
Application Debugging and Profiling

URL in the sending servlet. So, click on "setEndPoint" method and in the url: box, set the location of the rpcrouter servlet (i.e., <http://wsc1.washington.ibm.com:8887/PolicyIVP2/servlet/rpcrouter>) and then click the "Invoke" button. Now, click on "getEndPoint" method, press the "Invoke" button to see if you typed the URL correctly and you should see:



Now, click on "tran1" method and enter "cmp", 123 and 456 as shown below.



Now, press the "Invoke" button and in the fullness of time you will notice the action is carried out, the Result window will be cleared and the browser will report "done". If the action fails you will get an error message in the "Result" window. (Providing positive feedback is left as a "user exercise"). If you want to be certain everything worked correctly, look in the SYSPRINT data set for the WAS server region (i.e., WSPMR1S). To see an error message, change the port number to something invalid using the setEndPoint() method and invoke the tran1 method.

At this point you are ready to proceed with using the distributed debugger.

Remote application debugging

Remote application debugging will be done using the WSAD tool.

WSAD

Using the WSAD distributed debugging capability on the Policy IVP is amazingly simple.

Host:

All that is required on the host side is to set a single environment variable: JVM_DEBUG_PORT in the current.env file for your server specifying the port you want to use (have to check with the guardians of TCPIP again). For this example, the current.env file for the WSPMR1 server was edited and JVM_DEBUG_PORT=8001 was included. The server was then restarted.

Look in the JES messages for the server region you should see the following message:

```
BBOU0161I SHASTA RUNTIME FUNCTION LoadAndInitVM DETECTED THAT  
The jdwp port is set to 8001.
```

The WAS for z/OS server is now ready to have the distributed debugger connect to it on port 8001.

Workstation:

Prior to using the debugger, WSAD needs to be invoked using the supplied workspace. Edit the WSAD-IVP.bat file in the directory where you unzipped PolicyIVP V2.zip file. The file contains the following:

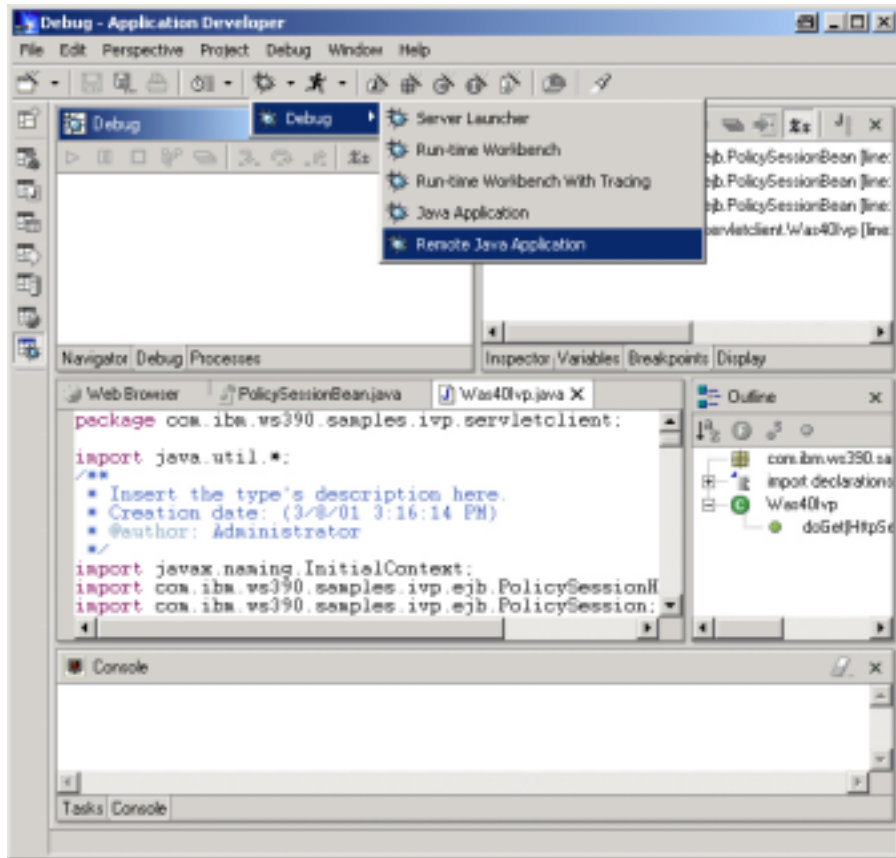
```
Rem  
Rem Start WSAD using the workspace for this IVP2 project.  
Rem  
"D:\PROGRA~1\IBM\APPLIC~1\wsappdev.exe" -data d:\zzz\IVP401\workspace
```

You must update location/path of wsappdev.exe AND set the correct location of the workspace directory. Once you have made these changes, invoke the .bat file to start WSAD using this workspace.

Now, all that remains is to inform the WSAD debugger of the location and port for the WAS for z/OS server whose application is to be debugged.

In the WSAD tool, change the perspective to the Debugger Perspective. Select the small "down arrow" next to the debug icon → Debug → Remote Java Application. You will be presented with the following popup window sequence:

WebSphere for z/OS Application Debugging and Profiling



Clicking on "Remote Java Application", you may receive the following popup:



If you do, select PolicyIVP2 and press the "Next" button. At this point you will be presented with a popup that allows you to specify the host and port number that should be used for this debugging session. For this

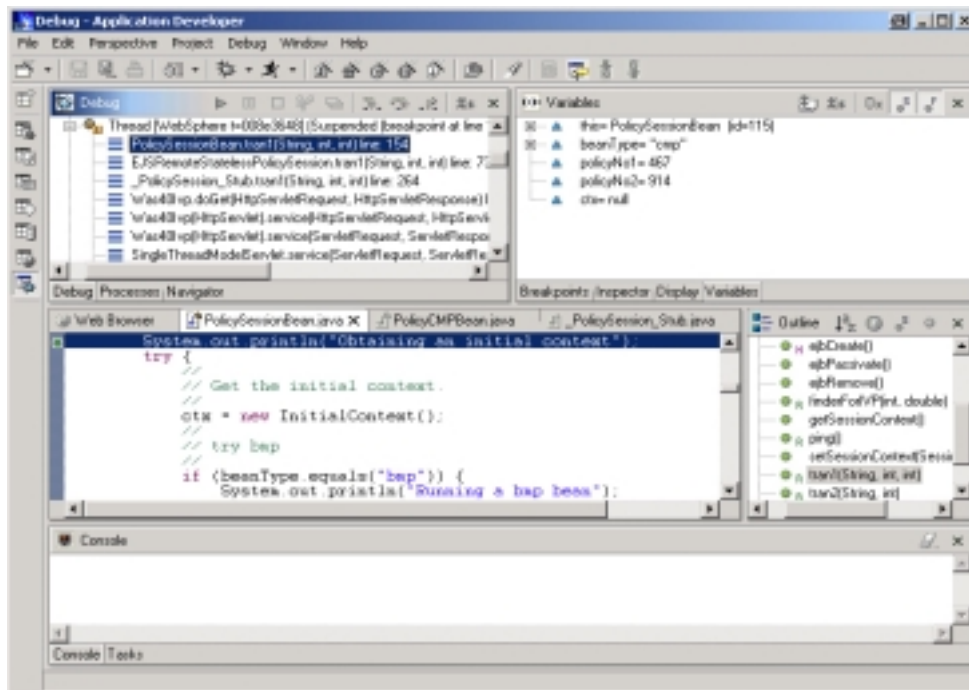
WebSphere for z/OS Application Debugging and Profiling

example, the host name is: wsc1 and the port number is: 8001 (the same port as specified in the current.env file with the JVM_DEBUG_PORT=8001 setting).



Click "Finish" and the debugger will connect to the remote JVM. That is all there is to it. You are ready for a debugging session.

You will notice breakpoints are already set in the application. There are breakpoints in the servlet and the session bean. Start an instance of the browser of your choice and select the URL associated with the Policy IVP web application (i.e., <http://wsc1.washington.ibm.com:8887/PolicyIVP2/cebit.html>). When you receive the main panel in your browser, enter the policy numbers, say '467' and '914', select 'CMP' option and press the "Submit" button. In the fullness of time, the first breakpoint in the doGet() method on the Was40Ivp servlet will hit. You can examine the local variables. Then press the "Resume" icon (or F8) to proceed to the next breakpoint in tran1() method of PolicySessionBean. You will see something like:



WebSphere for z/OS
Application Debugging and Profiling

Press the "Resume" icon again to proceed to the second breakpoint in this method. Pressing the "Resume" icon once again will get you to the first breakpoint in tran2() method of the PolicySessionBean. Finally, pressing the "Resume" icon yet again will allow the transaction to run to completion and you should get the normal successful completion text on your browser.

Feel free to set additional breakpoints and re-run the application. Once you are finished, remove the JVM_DEBUG_PORT environment variable from the current.env file for the server and proceed to the next section.

Profiling the Application

There are at least two choices of profiling tools, in this paper we will use two: the WSAD tooling and the Jinsight tooling. There are tradeoffs in using either one.

- ◆ The profiling tool of WSAD communicates with the Remote Agent Controller daemon, which must be active on the z/OS system. This daemon uses port 10002.
- ◆ Jinsight collects all of trace data in a file on the z/OS system.
- ◆ Jinsight usage requires the controlling web application be installed in the server.
- ◆ The visualization tools are somewhat different in capability.

There are circumstances where one or the other might be more applicable, hence both will be discussed.

WSAD Profiling:

Host:

The most difficult part is finding the Remote Agent Controller (RAC) code for z/OS. In the WSAD distribution is a directory called: RAC_install that contains a number of directories. The directory in which we have interest is: RAC_install/zOS. This directory contains the z/OS specific code in a pax file (i.e., ibmrac.os390.pax.Z). Also in the WSAD distribution is documentation in the file: readme/ws/readme_profile.html file. This file will take you through the installing, customization and running of the remote agent controller on the z/OS system and should be considered the definitive source for information.

Basically, one must FTP the file ibmrac.os390.pax.Z into the chosen directory on the z/OS system in BINARY format. Then unpx the file with the following command:

```
pax -rvzf ibmrac.os390.pax.Z .
```

The documentation suggest unpxing the file into usr/lpp directory, for this exercise the file was unpxed into /shared/lib directory using a userid with a UID=0. This action produces a number of directories and files. At this point, some manual work is involved:

1. You must change to the IBMRAC/lib directory and change the file attributes of all the .so and .dll files to 755. If you neglect to do this task, the JVM will not successfully load the profiling code and the server will fail to initialize. Note: this may not be the case in later service levels, check before blindly changing the file attributes.
2. You must customize the IBMRAC/config/ serviceconfig.xml file if you did not place the IBMRAC directory in the default location (i.e., /usr/lpp) **OR** if the Java SDK is installed in a directory other than the default directory (i.e., /usr/lpp/java/IBM/J1.3).
3. You must customize the IBMRAC/bin/RASStart.sh script if you did not place the IBMRAC directory in the default location (i.e., /usr/lpp).

After tailoring of the configuration file and startup script, invoke the RASStart.sh script to start the RAC daemon. The RAC daemon must be running prior to starting a WAS for z/OS server address space which will have an application to be profiled.

Note: At the level of RAC code during this exercise, it was necessary to run the RASStart.sh script from a userid with UID=0 in order for the daemon to write into the servicelog.xml file.

WebSphere for z/OS
Application Debugging and Profiling

Now, update the following environment variables for the server. The following variables were set/updated in the current.env file for the WSPMR1 server:

```
JVM_EXTRA_OPTIONS=-XrunpiAgent
JAVA_COMPILER=1
LIBPATH=/shared/lib/IBMRAC/lib:/usr/lpp/db2/db2710/lib:/usr/lpp/java2/J
1.3/bin:/usr/lpp/java2/J1.3/bin/classic:/usr/lpp/WebSphere401/lib
```

Two new environment variables must be set.

- ◆ The first allows the profiling code to be invoked by the JVM.
- ◆ The second variable specifies the JIT is to be disabled (If you fail to do this you will get a popup regarding a stack underflow when the profiling is started, while not fatal it is a nuisance).

The LIBPATH variable string must be updated to include the path to libpiAgent.so, which in our example is the /shared/lib/IBMRAC/lib directory.

Make certain the JVM_DEBUG_PORT is not specified.

At this point, restart the server in order for the new environment variables to take effect. With the JIT disabled, initialization and subsequent interactions will be slower than normal. There are no messages produced by the profiling code in the joblog to indicate that profiling agent code is enabled. However, if you issue a netstat command, among the output you will see something like the following:

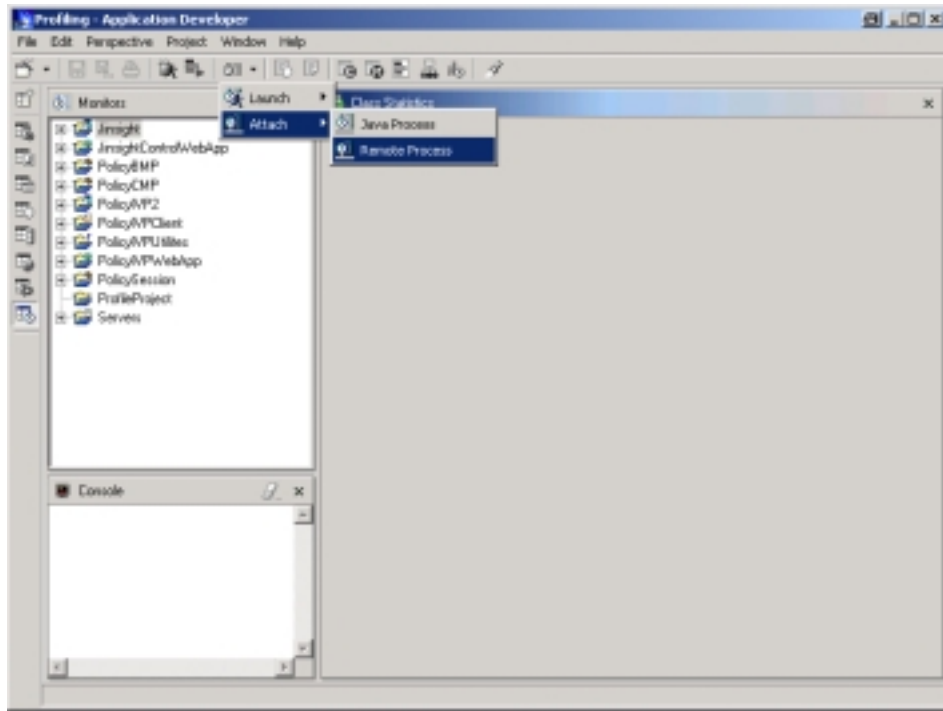
```
. . . .
MCCOX6  000138E5 0.0.0.0..10002      0.0.0.0..0      Listen
MCCOX6  00013EE6 9.82.93.37..10002      9.82.93.37..1952  Estabsh
. . . .
WSPMR1S 00013EE5 9.82.93.37..1952      9.82.93.37..10002  Estabsh
. . . .
```

Port 10002 is being listened on by the RAC daemon, which was started by userid MCCOX. The RAC Daemon is also in session with the server region (i.e., WSPMR1S) as well. All that remains is getting the WSAD profiling environment connected to the RAC daemon to begin profiling the Policy IVP application.

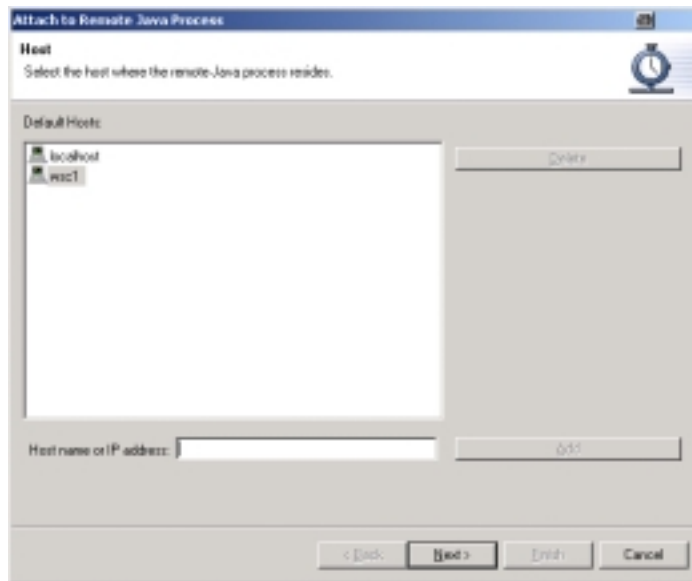
Workstation:

In the WSAD application, open a profiling perspective, then pres the small "down arrow" next to the "stopwatch" icon → Attach → Remote Process. You should see the following sequence of popup windows:

WebSphere for z/OS Application Debugging and Profiling

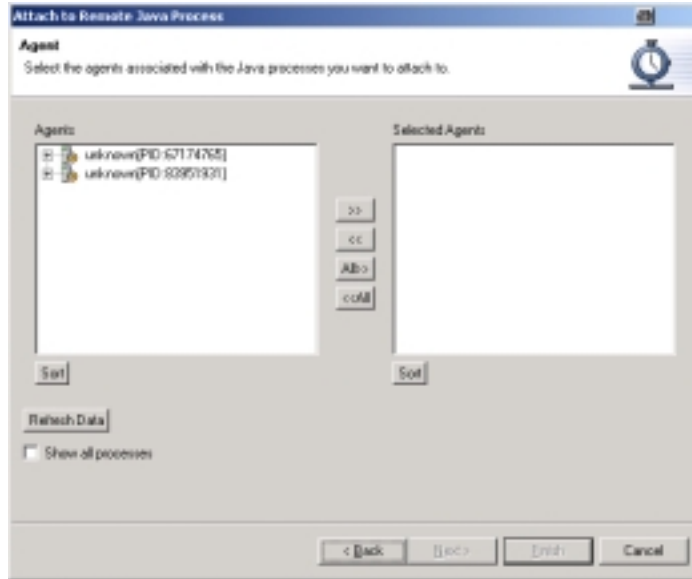


Click on "Remote Process". At this point you receive a pop-up window, enter the host address, press "Add" button and you will see the following:



Press the "Next" button to get a popup like the following:

WebSphere for z/OS Application Debugging and Profiling

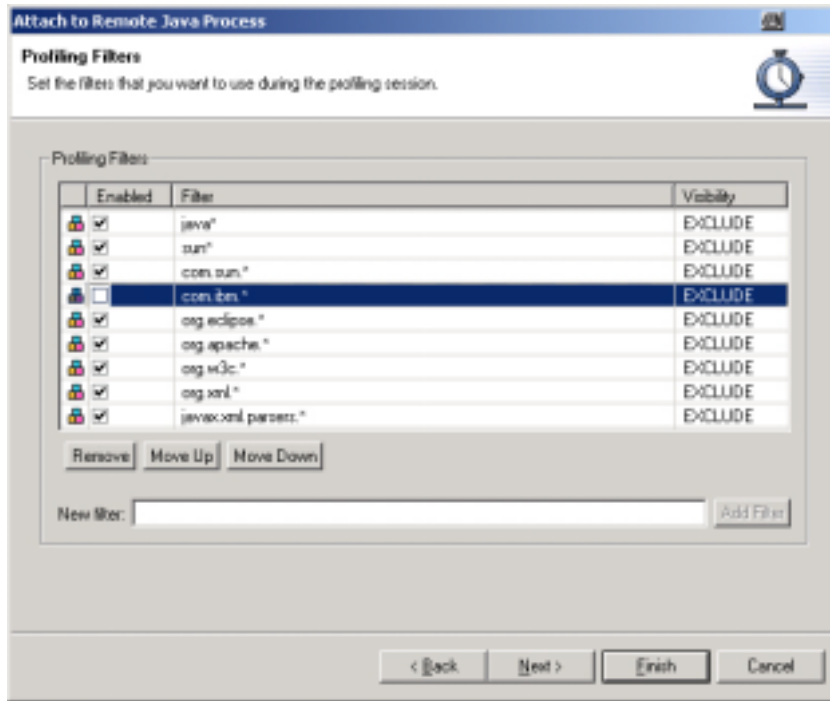


The question is: Which Agent do I select (If you only have one on your screen you an easy choice). In this case it appears that someone else is profiling an application and it is necessary to connect to the correct server. On the z/OS system, issue the ps command as follows:

```
:/shared/lib/IBMRAC/bin-> ps -Uwspmr1s
      PID TTY          TIME CMD
 16842920 ?           0:21 BBOSR
 67174765 ?           0:29 BBOSR
```

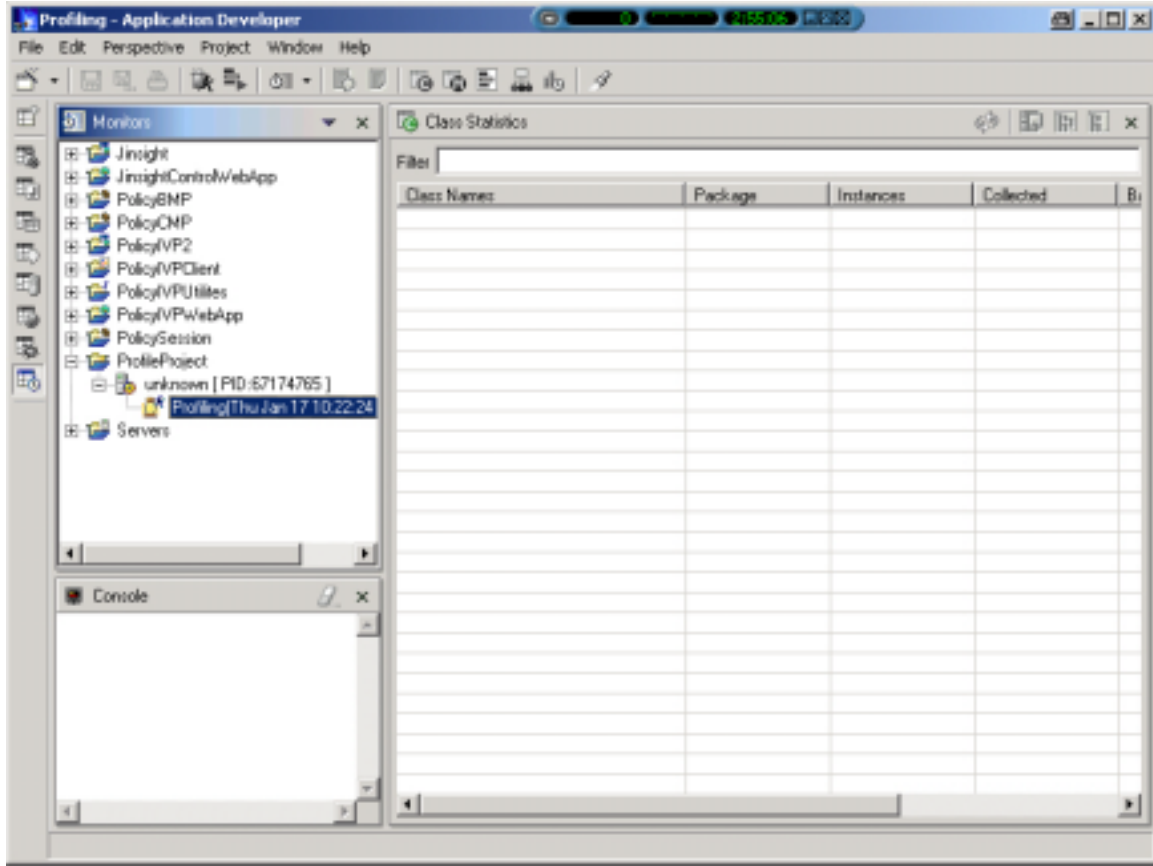
The server address space in which the Policy IVP application is running is: WSPMR1S which is associated with userid: WSPMR1S, hence the command syntax. Now, you know the agent to select is: 67174765. So highlight the correct agent and press the ">>" button, then press the "Next" button. In this pop-up (not shown), you will get the opportunity to name the Monitor, do so and then press the "Next" button to get the following screen:

WebSphere for z/OS
Application Debugging and Profiling



If you take the defaults, you will not be able to profile the PolicyIVP application as all of its packages start with com.ibm.*, so uncheck the box and press the "Finish" button. At this point, you will see the following profiling perspective pane in WSAD:

WebSphere for z/OS Application Debugging and Profiling



If you issue a netstat command on the z/OS system, you will see something like the following:

```
. . .
MCCOX6  00013EF2  9.82.93.2..10002      9.15.9.151..1297      Establish
MCCOX6  000138E5  0.0.0.0..10002      0.0.0.0..0           Listen
MCCOX6  00013EE6  9.82.93.37..10002   9.82.93.37..1952     Establish
. . .
```

Notice that your workstation is now in session with the RAC daemon along with the server address space. You are ready to profile the application.

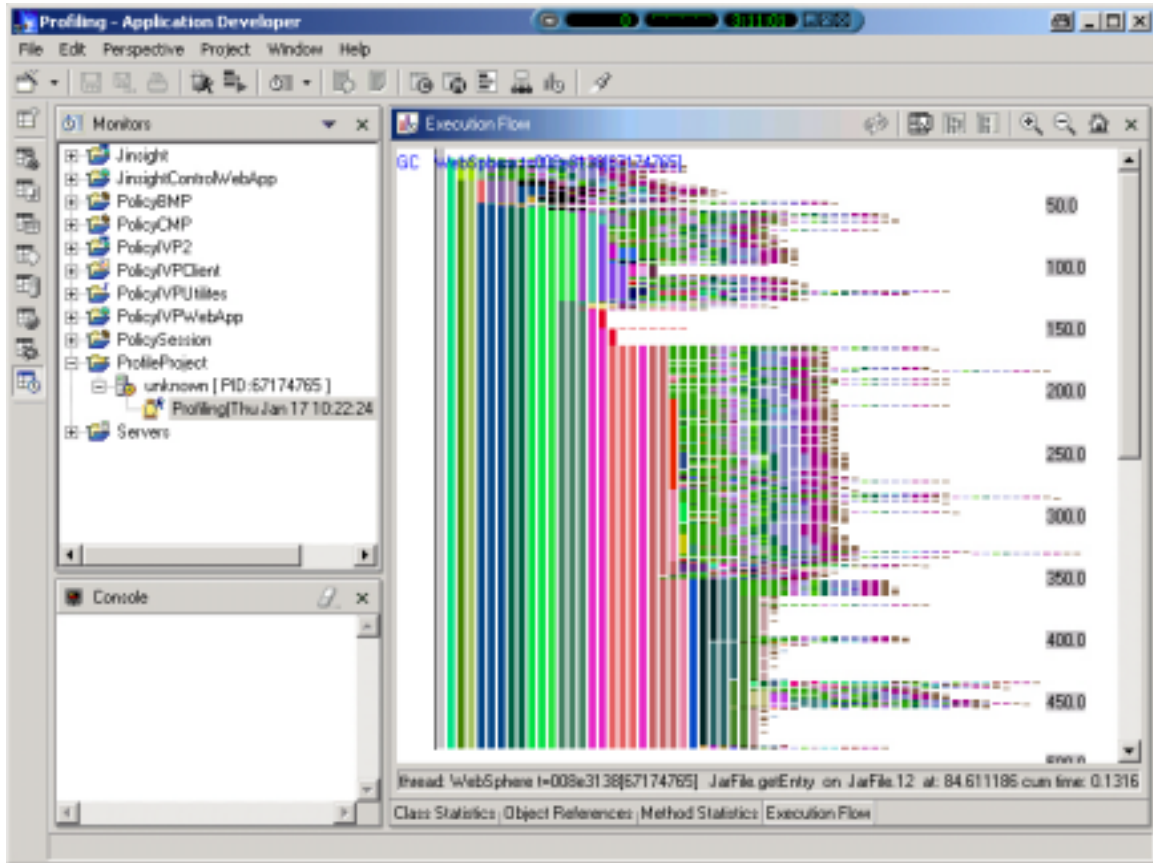
The general idea is to invoke the transaction or use case that you wish to profile once prior to enabling the monitoring, just so the application server can load all the required classes. There is not a lot of reason to profile application start up. Start another instance of the web browser of your choice and invoke a URL to drive the servlet (i.e.,

<http://wsc1.washington.ibm.com:8887/PolicyIVP2/PolicyServlet?policyno1=334&policyno2=543&beauty=cmp>). Once the response is returned, you are ready to profile your application code.

Start the monitoring of the server by clicking mouse button '2' on the highlighted profiling session, then select "Start Monitoring" option (not shown). Now invoke the application using the web browser of your choice. For example use a URL to drive the servlet without the using input cebit.html page (i.e., <http://wsc1.washington.ibm.com:8887/PolicyIVP2/PolicyServlet?policyno1=467&policyno2=914&beauty=cmp>). Once the response has been returned, press the "Dump objects and References" button, then select then select the active profiling session, click mouse button '2' and select "Stop Monitoring" (not shown). The data has been collected, there is no need to continue monitoring until you wish to look at a different transaction or use case.

WebSphere for z/OS Application Debugging and Profiling

Now choose one of the profiling visualization views, for example execution flow, click mouse button '2' in the frame and select "Update views" to get current data from the RAC daemon. You will see a pop-up window indicating data is flowing, and then you will get a screen similar to the following:



There are other views of the profiling session that you can examine. Feel free to do so.

Once you have finished collecting the profiling information, select the active profiling session, click mouse button '2' and select "Detach from Agent" and confirm with "Yes" on the subsequent pop-up. At this point your profiling session is over and you no longer have a session with the RAC daemon from your workstation. At this point you should update the current.env file for the server and remove:

```
JVM_EXTRA_OPTIONS=-XrunpiAgent  
JAVA_COMPILER=1
```

You can leave the LIBPATH as it is, the profiling code will not get invoked as long as the JVM_EXTRA_OPTIONS statement is removed. Once the current.env changes are made restart the server without the profiling agent code being active.

If you are finished profiling with WSAD on this system, you should terminate the RAC daemon using the RASStop.sh script.

Jinsight Profiling:

Host:

First you need to install the Jinsight 2.1 code on the z/OS system. The code is located on your workstation, (i.e., D:\Program Files\jinsight2.1) as the result of unzipping the package previously downloaded from <http://www.alpha-works.ibm.com>. In the directory where the jinsight code is installed, you will see a .tar file: jinsight2.1-os390-java2.tar. This file contains the z/OS specific code that must be installed on the z/OS system to enable Jinsight profiling. FTP this file in BINARY mode to the z/OS system and untar it into the directory of your choice. In this example, the file is untarred into the /shared/lib directory using a userid with a UID=0. This action will create a directory named jinsight2.1 and appropriate contents.

NOTE: After you have untarred this file, you will have to set the file ownership values appropriately as the tar file was not built with the correct options.

All that remains is to update the environment variables for the WSPMR1 server to allow the Jinsight profiler to collect data. The following variables must be set/updated in the current.env file:

```
JVM_EXTRA_OPTIONS=-XrunjinsightPA
JINSIGHT_TRACEFILE_NAME=/tmp/WSPMR1.jinsight.trace
LIBPATH=/shared/lib/jinsight2.1:/usr/lpp/db2/db2710/lib:/usr/lpp/java2/
J1.3/bin:/usr/lpp/java2/J1.3/bin/classic:/usr/lpp/WebSphere401/lib
```

Two new environment variables are set.

- ◆ The first allows the Jinsight profiling collection code to be invoked by the JVM.
- ◆ The second variable specifies where the trace file is to be written. (Since you cannot specify an address space qualifying variable in the name of this file, it is important that only one server region write to into the trace file. Hence the recommendation to restrict the number of servers to one for this exercise.)

The LIBPATH variable string must be updated to include the path to libjinsightPA.so, which in our example is: /shared/lib/jinsight2.1 directory.

Make certain the JVM_DEBUG_PORT is not specified.

Restart the server so the new environment variable settings will take effect. The server is ready to collect profiling data when instructed to do so using the Jinsight controller web application. Look at the SYSOUT DD statement for the server region and you will see the following:

```
*****
Jinsight2.1, build 20010629
Jinsight Profiler is Licensed Materials - Property of IBM
Copyright (c) IBM Corp. 2000
IBM is a Trademark of International Business Machines
*****
Authors: Jeaha Yang      Fereydoun Maali
*****
OS/390 : use CTRL-V for tracing options
*****
```

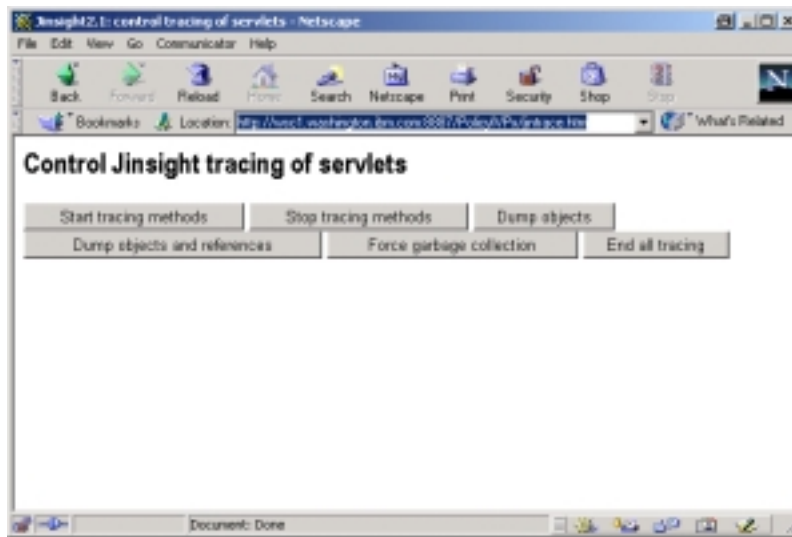
The server is ready to run the application and collect trace data if instructed to do so.

Workstation

The general idea is to invoke the transaction or use case that you wish to profile once prior to collecting the profiling data, just so the application server can load all the required classes. There is not a lot of reason to profile application start up. Start an instance of the web browser of your choice and invoke a URL to drive the servlet (i.e.,

<http://wsc1.washington.ibm.com:8887/PolicyIVP2/PolicyServlet?policyno1=334&policyno2=543&beauty=cmp>). Once response is returned you are ready to profile your application code.

Using another instance of the browser of your choice invoke the URL for the Jinsight controller web application (i.e., <http://wsc1.washington.ibm.com/PolicyIVP2x/jintrace.htm>) and you will see the following:



From the Jinsight control panel press the "Start tracing methods" button. Now, run your transaction/use-case once (reload the URL from which you just drove the servlet in the other browser instance). Return to the Jinsight control panel, and press the "Dump objects and references" button. Then return to the Jinsight control panel and press the "End all tracing" button.

At this point your application profile trace file (i.e., /tmp/WSPMR1.jinsight.trace) will have quite a lot of data in it, usually millions of bytes. Now, use FTP to transfer this file in BINARY mode to your workstation for analysis. If you are finished profiling your application, remove the following environment variables from the current.env file:

```
JVM_EXTRA_OPTIONS=-XrunjinsightPA
JINSIGHT_TRACEFILE_NAME=/tmp/WSPMR1.jinsight.trace
```

You can leave the LIBPATH as it is, the profiling code will not get invoked as long as the JVM_EXTRA_OPTIONS statement is removed. Restart the server to remove the Jinsight profiling agent code.

Now you have data to analyze. First you must update the jinsight.bat file to specify the correct locations of Jinsight code, java, and the browser of your choice. In this example the contents of the file are:

```
@echo off
echo Visualizing Jinsight 2.1 trace data ...
echo Usage example:  jinsight trace.trc
rem
```

WebSphere for z/OS Application Debugging and Profiling

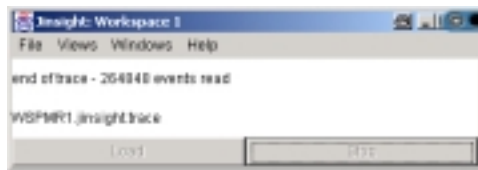
```
echo Edit these env vars if changes are needed:
set JAVA2_HOME=c:\progra~1\IBM\java13\jre
set JINSIGHT_HOME=d:\progra~1\IBM\jinsight2.1
set JINSIGHT_HEAP=512M
set JINSIGHT_BROWSER=c:\progra~1\netscape\communicator\program\netscape.exe
rem
rem Env vars being used are:
echo JAVA2_HOME=%JAVA2_HOME%
echo JINSIGHT_HOME=%JINSIGHT_HOME%
echo JINSIGHT_HEAP=%JINSIGHT_HEAP%
%JAVA2_HOME%\bin\java -version
%JAVA2_HOME%\bin\java -Xmx%JINSIGHT_HEAP% -classpath
%JINSIGHT_HOME%\jinsight.jar;. -DJINSIGHT_MAXHEAP=%JINSIGHT_HEAP% -
DJINSIGHT_BROWSER=%JINSIGHT_BROWSER% -DJINSIGHT_DOC=%JINSIGHT_HOME%\docs\
jinsight.main.Jinsight %1
```

The folding of the last line into three lines distorts the contents of the file. However you must set JAVA2_HOME, JINSIGHT_HOME, and JINSIGHT_BROWSER appropriately for your workstation. Set the JINSIGHT_HEAP to a large value, but one that can be supported by the memory on your workstation.

Once the jinsight.bat file is tailored, from the directory where jinsight is installed invoke the .bat file providing the name of the trace file residing on your workstation, for example:

```
D:\Program Files\IBM\jinsight2.1>jinsight d:\temp\WSPMR1.jinsight.trace
```

Press the "Load" button on the Jinsight application window to load the contents of the trace file. The jinsight visualization tools require substantial storage on the workstation, thus you probably should shutdown other processes (e.g., WSAD, Lotus Notes, VAJ, etc.) prior to loading the trace file. The Jinsight application will indicate the file has loaded. You can then click on "Views" to see any of the views of the application.



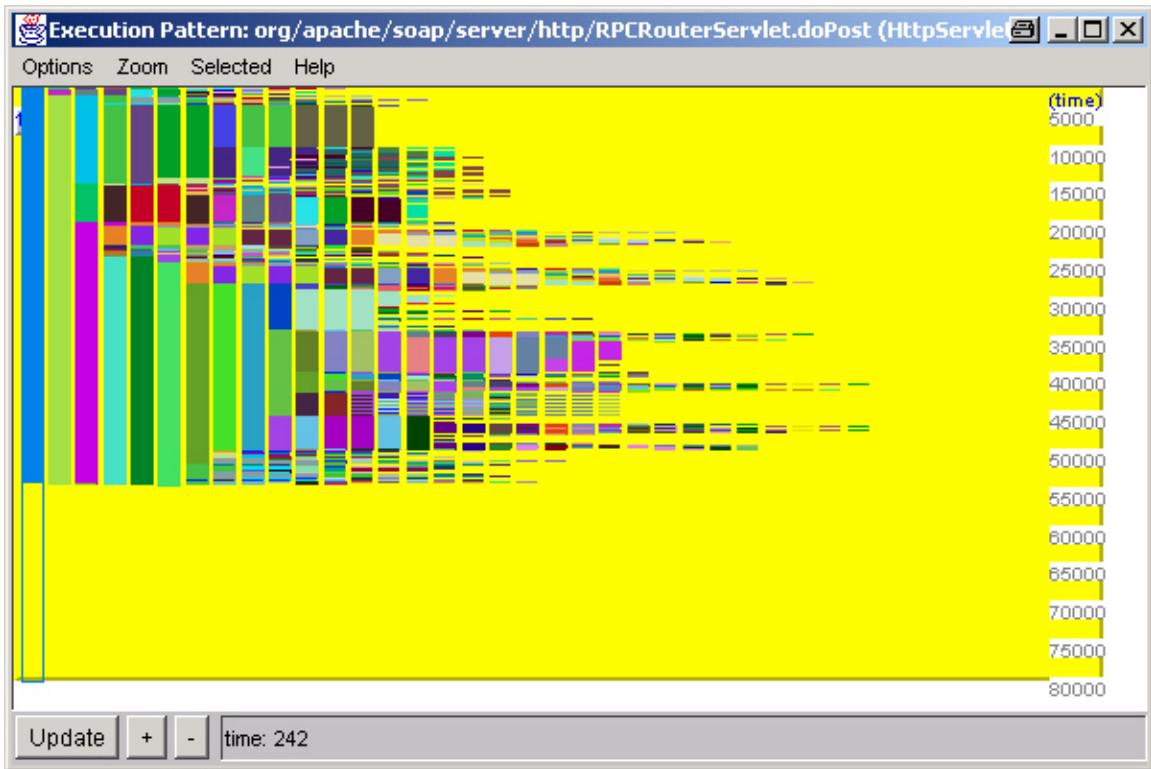
At this point you are on your own, you can use the entire suite of Jinsight visualization tools to analyze your application. The documentation for Jinsight is quite good, read it.

For example, here is a method level table representation from a trace of the web service that is included in the new version of the Policy IVP.

WebSphere for z/OS
Application Debugging and Profiling

class name	method name
com/ibm/servlet/engine/webapp/IdleServletState	service (StrictLifecycleServlet, ServletRequest, ServletResponse)
com/ibm/servlet/engine/webapp/StrictLifecycleServlet	_service (ServletRequest, ServletResponse)
com/ibm/servlet/engine/webapp/StrictServletInstance	doService (ServletRequest, ServletResponse)
javax/servlet/http/HttpServlet	service (ServletRequest, ServletResponse)
javax/servlet/http/HttpServlet	service (HttpServletRequest, HttpServletResponse)
org/apache/soap/server/http/RPCRouterServlet	doPost (HttpServletRequest, HttpServletResponse)
org/apache/soap/server/http/ServerHTTPUtils	readEnvelopeFromRequest (Document)
com/ibm/servlet/engine/invoke/CacheableInvocationContext	invoke (Object)
com/ibm/soap/providers/impl/WASStatelessEJBProviderImpl	invoke (SOAPContext, SOAPContext)
com/ibm/ws390/samples/ivp/ejb/_PolicySession_Stub	tran1 (String, int, int)
com/ibm/ws390/samples/ivp/ejb/EJSRemoteStatelessPolicySession	tran1 (String, int, int)
com/ibm/ws390/samples/ivp/ejb/PolicySessionBean	tran1 (String, int, int)
com/ibm/ws390/samples/ivp/ejb/_PolicyCMPHome_Stub	findByPrimaryKey (PolicyCMPKey)
com/ibm/ws/http/HttpRequest	readRequest ()

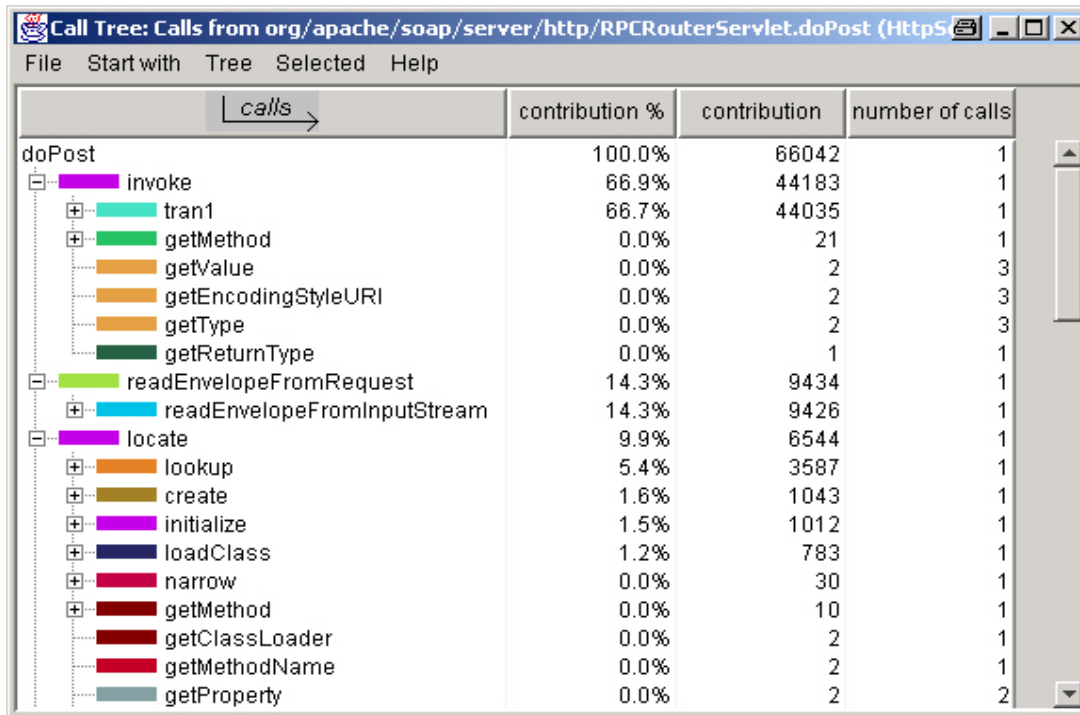
The processing part of interest is everything which happens in the doPost() method on the RPCRouterServlet. This is code that catches the inbound SOAP request and drives the tran1() method on the PolicySessionBean. You can drill down into the RPCRouterServlet.doPost() method to see the execution pattern:



If you move the cursor over the vertical bars, you will see the methods being invoked on the status bar at the bottom of the browser window.

WebSphere for z/OS Application Debugging and Profiling

If you drill down into the call pattern, starting with doPost(), you will see that 66% of the elapsed time was spent in the actual PolicySessionBean.tran1() method, while approximately a third of the elapsed time is spent in marshalling/de-marshalling the request, as seen in the following frame:



	contribution %	contribution	number of calls
doPost	100.0%	66042	1
invoke	66.9%	44183	1
tran1	66.7%	44035	1
getMethod	0.0%	21	1
getValue	0.0%	2	3
getEncodingStyleURI	0.0%	2	3
getType	0.0%	2	3
getReturnType	0.0%	1	1
readEnvelopeFromRequest	14.3%	9434	1
readEnvelopeFromInputStream	14.3%	9426	1
locate	9.9%	6544	1
lookup	5.4%	3587	1
create	1.6%	1043	1
initialize	1.5%	1012	1
loadClass	1.2%	783	1
narrow	0.0%	30	1
getMethod	0.0%	10	1
getClassLoader	0.0%	2	1
getMethodName	0.0%	2	1
getProperty	0.0%	2	2

Summary:

The facts are obvious:

One can indeed use the available distributed debugger and profiling tools on J2EE applications running in the WebSphere for z/OS runtime... and do so quite easily.