Developing Web Services with Eclipse and Open Source

Claire Rogers
Developer Resources and Partner Enablement, HP
February, 2004
Introduction

- Many companies investigating the use of web services
- Cost required to get started might pose a huge barrier
  - high-priced platforms may not be an option
  - teams may look to open source to get started
- Presentation objectives:
  - present a lifecycle for web services development
  - introduce open source tools for development
  - share key learnings in using some of these tools

The hope is that you will gain valuable knowledge to get started with web services in a cost-effective manner.
**Scenario**
- **HotSpell**, a new Internet-based startup company
- creating a weather forecast service for consumers
- has selected Java as underlying development platform
- wishes to expose application as a web service

**You will be led through the entire software lifecycle to develop, deploy, and manage the service using open source tools**
Initial Configuration: Selecting a Linux Distribution and Database

- **OS**
  - Why Linux?
    - viable alternative to a commercial operating system
    - cost effective
  - Why Debian Linux?
    - most vendor-neutral (“open source only” policy)
    - provides an easy-to-use interface for installing packages

- **Database**
  - PostgreSQL considered more robust in SQL support
  - MySQL designed for speed at the cost of features

- **Key learnings:**
  - provides both Unix and Windows look-and-feel
  - Linux is not a single development platform
  - installation made easier with aptitude and HP’s PTK
  - there are many different application packaging formats
Initial Configuration: Designing the Database

- Two simple tables were created to model the data
  - zipcode for city/state information
  - forecast for forecast information
- Data was loaded using the load data SQL command
- Permissions were added to allow user to query database tables
Developing the Java components
Selecting a Java Environment

- Linux JDK was required to run the application and tools
  - J2SE JDK from Sun (java.sun.com)
  - Blackdown JDK (www.blackdown.org)
  - BEA WebLogic JRockit (www.bea.com)

- JRockit selected for performance reasons
  - MxN threading model for Java threads
  - less memory and context switching
  - higher scalability for thread-intensive applications
Selecting a Java IDE

- **IDE**
  - Integrated Development Environment
  - provides tools to edit, compile, and debug applications

- **Several open source IDEs for Java available**
  - NetBeans ([www.netbeans.org](http://www.netbeans.org))
  - Eclipse ([www.eclipse.org](http://www.eclipse.org))

- **Eclipse** selected because of prior experience with tool
  - HP is Eclipse board member
  - Eclipse plug-ins available for OpenView

- **v.s.**
  - both open source
  - both provide Java-based IDEs
  - **Eclipse** is SWT-based
  - **NetBeans** is Swing-based
An Overview of Eclipse

Eclipse’s strength lies in its ability to easily integrate third-party tools into the development environment

Key features
- syntax highlighting editor
- incremental code completion
- source-level debugger
- class navigator
- file/project manager
- integration with source control systems
- task-oriented development through perspectives

The Eclipse Platform
Developing with Eclipse

- Installation and configuration was straightforward
  - downloaded the Linux version from www.eclipse.org
  - installed in /opt and updated PATH to include binary

- Key development steps:
  1. Create Project
  2. Create Java Classes
Creating the Java Classes

Weather.java

getWeather(String zip)

get zipcode information

get forecast for zipcode

create Forecast object

MySQL

zipcode

forecast

Forecast.java

String zip;
String city;
String state;
String date;
String forecast;
short hi;
short low;
byte precip;
Developing the Web service
The J2EE Web Container

- The web services runtime requires a J2EE web container
- We selected Tomcat
  - widely used open source servlet engine
  - default container for Apache products
- Installing Tomcat:
  - downloaded Tomcat 4.1.24 from jakarta.apache.org
  - configured environment variables
- Starting Tomcat:
  - startup scripts provided
  - Tomcat plug-in for Eclipse

A J2EE Container provides:
- lifecycle management
- security
- deployment
- runtime service
Tomcat Plug-in for Eclipse

- Starting, stopping and restarting Tomcat 4.x, 5.0.x, 3.3
- Registering Tomcat process to Eclipse debugger
- Creating a WAR project (wizard can update server.xml file)
- Adding Java Projects to Tomcat classpath
- Setting Tomcat JVM parameters, classpath and bootclasspath
- Exporting a Tomcat project to a WAR File
- Choosing Tomcat configuration file

http://www.sysdeo.com/eclipse/tomcatPlugin.html

Tomcat Plug-in

<eclipse_home>/plugins
The Web Services Container

- SOAP defines the XML message format for web services
- A Web Services Container:
  - manages the routing and receiving of SOAP messages
  - maps received SOAP messages to back-end components
  - provides tools for creating and deploying web services
- Apache Axis (www.apache.org/axis) was the open source platform chosen for this application
Designing the Web Service Interface

- **WSDL**
  - Web Services Description Language
  - defines the “signature” of the web service
  - XML-based, independent of platform

- **Two approaches for designing a WSDL**
  - design WSDL first, then map to business objects
  - have the WSDL be automatically generated from code

- **The “WSDL First” approach is usually recommended for complex document exchanges**
  - for our simple demo, only one method was exposed
  - we relied on Apache Axis tools to generate WSDL
Using Apache Axis

1. Create web service interface
   - Weather.java
   - Forecast.java
   - Run Java2WSDL
   - Weather.wsdl

2. Generate server side bindings
   - Run WSDL2Java
   - WeatherLocator.java
   - WeatherImpl.java
   - Deploy.wsdd

3. Package and deploy
   - Run javac and jar
   - Weather.jar
   - mysql-connector.jar
   - Copy to axis lib on Tomcat
   - AdminClient
   - Web service deployed
   - Weather.wsdl
   - Weather.java
   - Forecast.java
   - WeatherLocator.java
   - WeatherImpl.java
   - Deploy.wsdd
   - mysql-connector.jar
   - weather.jar
   - Web service deployed
Our Experience

- Existing code may not support web services model
- Apache Axis provided a sufficient development platform
  - mostly command-line as compared to other tools
- Generated server-side bindings were not “complete”
  - logic was added to invoke the original Java classes
- Deployment process was very straightforward
Eclipse Plug-ins for Web Services

- Improve WSDL Viewer for Eclipse
  - WSDL graphical view:
  - Automatic layout

- WSDL2JavaWizard
  - Wizard to generate client stubs
  - Invokes Apache WSDL2Java tool
    - Stubs, skeletons, data types
Creating an automated build process
Why Do You Need a Build Process?

- Tools are available for creating web services
  - for Apache Axis, process is mostly command-line
  - can be time consuming if components are rebuilt
- A build process can automate many of these steps
  - can greatly enhance developer productivity
- Consider an eXtreme Programming (XP) methodology
  - “continuous integration” – deploy early and often
  - single commands to build and test the web services
  - automated builds are conducted a few times a day
The Build Script

- A build project can contain multiple targets
  - a target represents a specific step in the build process
  - a target can have dependencies on other targets
- Targets contain tasks
  - creating, deleting, and copying files
  - compiling and packaging Java classes
- Apache Axis provides Ant tasks, e.g.:

```xml
<target name="wsdl2java">
  <axis-wsdl2java output="${proj.dir}" serverside="true" url="${proj.dir}/Weather.wsdl">
  </axis-wsdl2java>
</target>
```
Running the Build Script

Overall, use of Ant, combined with the integration into the Eclipse environment, provided us with an efficient mechanism to quickly build the various web services components.
Testing the Web service
Invoking the Service

- Client proxies isolate SOAP processing code
- Apache Axis automatically creates these components
- We had to write additional logic to use the proxy
Monitoring the Web Service

Apache Axis provides a TCP Monitor tool that monitors SOAP requests and responses

- getWeather SOAP request
- Forecast SOAP Response
- Proxy Configuration
Testing the Service

- Important considerations:
  - create graphical interfaces to test web services
  - design a test framework usable by non-developers
  - build tests into the development process early on
  - test security, reliability, interoperability, and scalability
  - consider the use of automated testing tools

Open Source Testing Tools
- **JUnit**: general framework for testing Java code
- **Grinder**: tool for load-testing web applications
- **Anteater**: Ant-based testing tool with SOAP support
- **PushToTest**: specifically targeted at web services
PushToTest TestMaker

- An open source web services testing tool
- Robust graphical environment and scripting language
  - tool can generate test case from given WSDL
  - scripts written used Jython (Python for Java)
  - comes with library to simplify creation of web services tests
- Allows you to test functionality and scalability of a web service
  - validate SOAP messages received
  - configure stress tests with multiple virtual clients

PushToTest TestMaker

www.pushtotest.com
Running the Test

```python
# create protocol for Axis servlet
protocol.setHost("localhost")
protocol.setPath("axis/servlet/AxisServlet")
protocol.setPort(8081)

# construct SOAP message with getWeather request
body.setTarget("weather")
body.getMethod("getWeather")
body.addParameter("zip", String, "80538", None)

# invoke service 100 times
totaltime = 0.0
for i in range(100):
    response = protocol.connect()
    totaltime += response.getTotalTime()

# print response
sprint "Avg. Response Time ="
sprint totaltime/100
sprint "ms to complete."
```

TestMaker Test Script

TestMaker Output
Using Eclipse to test the service

- **SOAP Raw Message Sender**
  - Eclipse plug-in available from [www.eclipse-plugins.info](http://www.eclipse-plugins.info)
  - View from which SOAP messages can be sent to a service
  - Useful for testing SOAP interoperability issues
  - Call any XML-based service over HTTP
Conclusion
Let’s Review…

- Select a Linux Distribution
- Select an Open Source Database
- Design the Database

- Select a Java Environment
- Select a Java IDE
- Develop the Java Application

- Select a J2EE Container
- Select a Web Services Container
- Design the WSDL
- Create the Web Service

- Design a Build Process
- Write the Build Script
- Run the Build

- Invoke the Service
- Monitor the Service
- Write Test Scripts
- Run the Test
Conclusion

Review:
- outlined a process for creating web services
- presented some open source tools that could be used

Our key learnings:
- were a few technical hurdles that had to be overcome
- we found these tools were a boost to our productivity
- surprised by the integration between tools
- process to locate/install Linux packages straightforward
- tools generally worked out of the box

Overall, the open source environment was very reliable, stable, and usable for building web services
What’s Next?

Monitoring, managing, and tracking the web services platform and web services

- HP is committed to adaptive management, automating the dynamic link between IT and business
- HP Openview offers products that integrate with Apache Axis:
  - OV SPI for Apache Axis captures information about the web service platform
  - OVTA will support Axis for diagnosing performance bottlenecks in web services
Call to action

- Linux and HP (www.hp.com/linux)
  - for more information about HP’s Linux strategy
- HP Dev Resource Central (devresource.hp.com)
  - for more information about web services development, web services management, and HP’s Eclipse initiatives
- Web Services Development and Open Source whitepaper (devresource.hp.com/drc/technical_white_papers/wsopensrc.pdf)
  - for an extended version of this presentation along with source code
- HP OpenView (www.openview.hp.com)
  - for more information about OpenView support for web services and open source