Under the Hood @WSO2

Paul Fremantle, CTO, WSO2
Paul Fremantle

- CTO and Co-Founder
- 10 years at IBM
  - STSM in WebSphere Development
- Infoworld CTO 25 in 2008
- VP, Apache Synapse
- Co-Chair, OASIS WSRX TC
Why look under the hood?

http://www.flickr.com/photos/kinghuang/3353421921/
Agenda

• WSO2
  – A little bit of history
    • Why we started?
    • Our vision?
  – Open Source at WSO2
  – Development model
    • Milestones
    • Build process
• Carbon
  – Front-end / Back-end separation
  – How to write a Carbon Component
  – cApp
  – A little bit about how Stratos fits
• Future work
Why did we start WSO2?

• The founders were (and still are):
  – Apache Members
  – Heavily involved in Open Standards and SOA
  – Believers in lightweight, lean software
  – Geographically distributed

• We set out to build a platform for building SOA applications
  – What we now call “Compositional Applications”
  – Combining existing apps, legacy systems, workflows, ESB, and new Java logic
  – Originally planned three products:
    • App Server, ESB, Business Process Server
Not just about the products
Good apps are *distributed* apps
Good apps are *connected* apps
Good apps are *open* apps
All good apps will one day be *legacy* apps
And about Open {source, development, standards,...}

Imitation is the sincerest of flattery.

Charles Caleb Colton, Lacon, volume I, no. 183 (1780 - 1832)
Applying Open Source principles to a company

• Mailing lists
  – Inclusive communications rather than exclusive
  – Most are public:
    • Architecture, Stratos-dev, Carbon-dev
  – Obviously not all!
    • Support-dev, bizdev, marketing are private

• Very flat technical structure
  – “Do-Ocracy”
  – We do have Senior Managers, Product Managers, Director of Engineering and CTO
  – Whole team contributes to all discussions
  – Annual offsite planning meeting with most of the team

• Open Development
  – Plans, builds and full source are in the open
  – Encourage patches and contribution from the community
Litmus test for an Open Source company

• Open license
• Open development
• Open community
• Open processes

Development process

• Heavily based on “Milestone” builds
  – Each milestone aims to go from a working system to a working system
  – With specific features and function added

• Based around on Carbon model (core vs features)

• Automated build and as much automated test as possible
  – Unit tests and Selenium-based testing
  – Full QA process before release
  – http://builder.wso2.org/~carbon/releases/carbon/3.0.0/
Carbon

http://www.flickr.com/photos/exquisitur/
Carbon Architecture

• “Eclipse for Servers”
  – Completely Modular Middleware
  – Based on the Equinox OSGi runtime
  – The most integrated composite application platform available

• Clean “front-end/back-end” separation
  – Every component has a core runtime, a clean SOA management interface, a well-defined front-end console component
  – All completely pluggable, versioned, etc

• Full dependency management
  – Hence full re-use

• Pluggable common core services:
  – Registry, Key Management, Identity Management, Clustering, Monitoring/JMX, Transports, etc
  – Cloud enabled
What is a Carbon Component?

- OSGi Bundles
- Lives in the Carbon framework
- Two aspects
  - Backend runtime
  - Frontend Console
- Uses Carbon services
  - via OSGi Service Registry
  - e.g. Registry, UserManager, Axis2, etc
p2 - Provisioning Platform

- p2 stands for “Provisioning Platform”
- p2 allows us to manage components in a controlled manner
  - Install, Uninstall, Revert
  - Based on a web or file based repository
    - Can be hosted internally for an organization
  - Three approaches:
    - Command line
    - Web console
    - Secure remote API
- Provision the middleware
- Early availability
  - Provision user applications - cApp
Patches

• Patches are inherently complex
  – Patch A may depend on Patch B may depend on Service Pack 2
  – p2 can solve this
• Since Carbon 3.0.0 patches are managed by p2
• Previous versions are patched in the more manual approach
• Patches are only available to WSO2 customers and licensed for use with supported production servers
Carbon Component Manager

Available Features

Find new features or updates to installed features in available repositories


- Show only the latest versions

Find Features

Install

Select all | Select none

<table>
<thead>
<tr>
<th>Name</th>
<th>Search</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSO2 Carbon - Logging Management Feature</td>
<td></td>
<td>3.0.0.SNAPSHOT</td>
</tr>
<tr>
<td>WSO2 Carbon - Max Module Feature</td>
<td></td>
<td>3.0.0.SNAPSHOT</td>
</tr>
<tr>
<td>WSO2 Carbon - Security Management Feature</td>
<td></td>
<td>3.0.0.SNAPSHOT</td>
</tr>
<tr>
<td>WSO2 Carbon - Service Management Feature</td>
<td></td>
<td>3.0.0.SNAPSHOT</td>
</tr>
<tr>
<td>Service Hosting Feature</td>
<td></td>
<td>3.0.0.SNAPSHOT</td>
</tr>
<tr>
<td>WSO2 Carbon - Statistics Feature</td>
<td></td>
<td>3.0.0.SNAPSHOT</td>
</tr>
<tr>
<td>WSO2 Carbon - Tools Feature</td>
<td></td>
<td>3.0.0.SNAPSHOT</td>
</tr>
<tr>
<td>WSO2 Carbon - JMS Transport Core Feature</td>
<td></td>
<td>3.0.0.SNAPSHOT</td>
</tr>
<tr>
<td>WSO2 Carbon - Mail Transport Core Feature</td>
<td></td>
<td>3.0.0.SNAPSHOT</td>
</tr>
<tr>
<td>WSO2 Carbon - Transport Management Feature</td>
<td></td>
<td>3.0.0.SNAPSHOT</td>
</tr>
</tbody>
</table>

© WSO2 Inc.
What do you need to know to add to Carbon?

- Java – to write some code
- Carbon
  - Understand the architecture and the usage
- Apache Axis2
  - enough to create a service and consume it
- JSP
- XML
- SVN
  - To check out the code
- Maven
  - To build the code

- Plus a little OSGi and a little p2
  - OSGi is useful for:
    - Getting OSGi services (e.g. Registry) injected into your code
    - Handling activation of bundles
Trying to understand the big picture

Custom Client

Browser

Console

Server

Repository

Aggregate Feature

Front end feature

Back end feature

© WSO2 Inc.
How to get started writing a Carbon Component

- Look at some sample code 😊

- Write a back-end (BE)
  - Some actual server logic plus an admin service to manage it
  - Add some special sauce into the services.xml to let us know it’s an admin service
  - component.xml ← configures permissions

- Write a front-end (FE)
  - Admin service client – standard Axis2 WSDL client
  - JSP
    - Helper code sets correct security
  - component.xml
    - Defines the menu structure to place the JSP in the right place
<serviceGroup>
  <service name="MyCCAdminService" scope="transportsession">
    <schema schemaNamespace="http://fremantle.org/xsd"
      elementFormDefaultQualified="true"/>
    <transports>
      <transport>https</transport>
    </transports>
    <parameter name="ServiceClass">org.fremantle.CCService</parameter>
    <parameter name="AuthorizationAction">/permission/protected/configure/logging</parameter>
  </service>
  <parameter name="adminService">true</parameter>
  <parameter name="hiddenService">true</parameter>
  <parameter name="superTenantService">true</parameter>
</serviceGroup>
<component xmlns="http://products.wso2.org/carbon">
    <ManagementPermissions>
        <ManagementPermission>
            <DisplayName>Configure</DisplayName>
            <ResourceId>
                /permission/protected/configure
            </ResourceId>
        </ManagementPermission>
        <ManagementPermission>
            <DisplayName>Logging</DisplayName>
            <ResourceId>
                /permission/protected/configure/logging
            </ResourceId>
        </ManagementPermission>
        <ManagementPermission>
            <DisplayName>Logs</DisplayName>
            <ResourceId>
                /permission/protected/monitor/logging
            </ResourceId>
        </ManagementPermission>
    </ManagementPermissions>
</component>
A little bit of OSGi

• Back end services can access the Carbon Core services as OSGi Services
• These get injected into your classes uses OSGi decoration
• E.g. RegistryService:

```java
/**
 * @scr.reference name="registry.service"
 * interface="org.wso2.carbon.registry.core.service.RegistryService"
 * cardinality="1..1" policy="dynamic" bind="setRegistryService"
 * unbind="unsetRegistryService"
 **/
private static RegistryService registryServiceInstance;
protected void setRegistryService(RegistryService registryService) {
    registryServiceInstance = registryService;
}
protected void unsetRegistryService(RegistryService registryService) {
    registryServiceInstance = null;
}
```
A little bit more OSGi

- Bundles can be “activated” automatically by OSGi
  - This is controlled by the bundles.info file
  - `CARBON_HOME/repository/configuration/org.eclipse.equinox.simpleconfigurator`

- p2 automatically checks if the feature’s bundles need activation and if so updates this file correctly

- So your BE service can auto-start when the server starts
  - Dynamically calculates startup order so that dependencies are started first
Front-end

![WSO2 Management Console](image)

<table>
<thead>
<tr>
<th>Server</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>192.168.1.67</td>
</tr>
<tr>
<td>Server Start Time</td>
<td>2010-07-22 13:24:01</td>
</tr>
<tr>
<td>System Up Time</td>
<td>0 day(s) 0 hr(s) 12 min(s) 17 sec(s)</td>
</tr>
<tr>
<td>Version</td>
<td>${product.version}</td>
</tr>
<tr>
<td>Repository Location</td>
<td>file:/WSO2CA~1.0/bin/ /repository/deployment/server/</td>
</tr>
</tbody>
</table>

**Operating System**
- **OS Name**: Windows 7
- **OS Version**: 6.1

© WSO2 Inc.
FE component.xml

```xml
<component xmlns="http://products.wso2.org/carbon">
  <menus>
    <menu>
      <id>my_menu</id>
      <i18n-key>log-admin.logs</i18n-key>
      <i18n-bundle>org.fremantle.i18n.Res</i18n-bundle>
      <parent-menu>configure_menu</parent-menu>
      <link>../my-admin/my-admin.jsp</link>
      <region>region1</region>
      <order>12</order>
      <style-class>monitor</style-class>
      <icon>../log-admin/images/log.gif</icon>
      <require-permission>
        /permission/protected/configure/logging
      </require-permission>
      <require-super-tenant>true</require-super-tenant>
    </menu>
  </menus>
</component>
```

The XML has had whitespace added for presentation.
JSP Snippets

• Import the Carbon Taglib
  `<%@ taglib uri="http://wso2.org/projects/carbon/taglibs/carbontags.jar" prefix="carbon" %>`

• Import the Axis2 client for the Admin Service
  `<%@ page import="org.wso2.carbon.logging.admin.ui.LoggingAdminAdminClient" %>`
More JSP

<%  
String backendServerURL = CarbonUIUtil.getServerURL(config.getServletContext(), session);

ConfigurationContext configContext = (ConfigurationContext) config.getServletContext().getAttribute(CarbonConstants.CONFIGURATION_CONTEXT);
String cookie = (String) session.getAttribute(ServerConstants.ADMIN_SERVICE_COOKIE);
LoggingAdminClient client; 
try {
    client = new LoggingAdminClient(cookie, backendServerURL, configContext);
globalLogData = client.getSysLog();
} catch (Exception e) { ...} %>
Even more JSP

- i18n
  `<fmt:bundle
    basename="org.wso2.carbon.logging.admin.ui.i18n.Resources">

- Breadcrumbs
  `<carbon:breadcrumb
    label="logging.management"
    resourceBundle="org.wso2.carbon.logging.admin.ui.i18n.Resources"
    topPage="true"
    request="<%=request%>"/>
Finally actually do something!

```java
<% String[] logLevels = client.getLogLevels(); %>
```
Build process

1. Write the components = bundles
   - e.g.
     - Service component (BE)
     - Admin UI component (FE)
     - Maybe a “User” UI (FE) too
   - Use maven bundle plugin to build the bundles

2. Build the feature
   - Use Maven p2 feature plugin
   - Example POMs here:

3. Finally build the p2 repository
   - Maven POM to build the p2 Repository
Some useful resources

• How to develop OSGi Bundles using the Maven Plugin
• Getting Started with WSO2 Carbon
• How to build a UI component for Carbon
• Maven POM to build a p2 Repository
Its not *that* complicated :-)

http://www.flickr.com/photos/alebonvini/
Interested?

http://www.flickr.com/photos/aussiegall/
WSO2 cApp
cApp

• Extending p2 and OSGi to customer code

• A deployment model plus tooling
  – Takes care of bundle-ing for you
  – Manages dependencies
  – Topology concept to capture compositional apps across machines

• Still in development but Studio builds are available
Carbon Apps (cApps)

- A cApp is a repository of features
- Each feature is a deployable OSGi bundle containing a part of an application
  - E.g. ESB sequence, Services, BPEL flow, etc
- The cApp contains a “logical topology”
  - Mapping from app components to roles
- An automated build/deploy process adds a “physical topology”
  - Mapping into physical Carbon VMs
- The application features will have dependencies on the runtime components they run on:
  - Deploying an ESB sequence will ensure there is ESB code to run it on
  - Future!
Carbon Studio M12
How about Stratos?

http://www.flickr.com/photos/galego/
Tenant Programming Models

- Sub-Tenant Programming Model
  - Business as usual
  - Aka the Carbon Programming Model

- Super-Tenant programming model
  - How to write an application that can be deployed to all tenants
  - How to write a multi-tenant app
  - The basis of building a new Stratos Service

- This is still in “research” but we are willing to work with early adopters
Futures

- JCR Repository and WebDAV support
  - Based on the same core repository as the Governance Registry
- Message Broker (JMS broker)
  - Integrated into Carbon
  - Security, Logging, and Admin/Monitoring
  - Can simplifies deployment topologies compared to a standalone broker
- Event Server
  - Web and Web Service support for queues and topics
  - Complex Event Processing support
- cApp and Studio
- More Stratos services
Thank you