Fall 08

# Integration of AWS, BBS, and TITAN BCW

Benjamin Rockstroh

**This document describes the creation and testing of stub services for the BBS Service and AWS Service along with the TITAN BCW. The technologies used in this process are as follows, Axis2, WSDL, JAVA, Apache Tomcat, and Netbeans.**

## Purpose:

The purpose of this project is to test the interaction of the BBS Service, AWS Service and the TITAN BCW 2.0.

## Current Situation

Current TITAN based work on the BBS Service includes a 1) Stub service hosted on Tomcat, with a WSDL description. 2) Unit Action event that queries the stub service.

### Stub BBS Service

Created to test integration with TITAN BCW 2.0. The service receives a String representing a DSG and provides back a String stated "". Though this is a simple service, providing no real information in itself, since the BBS Service is

### Stub AWS Service

Created to test integration with TITAN BCW 2.0. The service receives a String representing an area or path and responds about information on the path. The current service queries the BBS Service and then waits a random about of time before it responds.

The AWS waits to simulate the passing of events in the field, since the AWS Service is designed to continually monitor an area or path, and respond back with changes. The call to the BBS Service is necessary since it is expected the AWS Service will utilize the BBS Service for determining different indications, warnings, or alerts.

Both JAX-WS synchronous and asynchronous methods could be used for this service, though the choice would dictate the design of the client.

#### Synchronous

Client would receive a response object, which it would poll to determine when the service completed the request.

#### Asynchronous

Client would provide the service with a callback handler.

## Setup

The BBS Service and AWS Service are hosted on an Apache Tomcat 6.0.2 server running on OS X 10.6. The TITAN BCW is running on the same machine, though connection to the BBS Service and AWS Service with a HTTP SOAP2 request. The system is connected to a local Intranet with a 192 IP address.

The TITAN sweet is designed to run over a JMS bus. It is possible to set these services up to listen and respond via this bus. A quick Google search shows this has been done before, though should be tested for this environment.

## Generate Service

Two separate approaches were used in creating the BBS Service and AWS Service stubs.

### WSDL to JAVA

The first approach was to create the WSDL file by hand, and use the WSDL to JAVA generator AXIS2. Both command line and a Netbeans plug-in were used, and both were found to be fairly simple.

### JAVA to WSDL

The second approach was to write a JAVA class file with methods for each one of the service calls. Then the Netbeans plug-in for AXIS2 was used to generate a WSDL and service bindings for the class. This approach provided a well-formatted WSDL file, and did not require any work with XML. Since working with XML can be rather cumbersome this approach was preferred over creating the WSDL by hand.

## Plug-in

Currently the TITAN BCW does not provide right click interface for paths or units. As a result I have created a BBS Service action to call the BBS Service with a Unit's DSG. The AWS Service is currently not integrated with the BCW.

## Service Descriptions

### Battle Book Service

| Methods | Input | Response |
|---|---|---|
| setUnitInformation | dsg:STRING<br>info:STRING | A String representing the information for the unit with the corresponding DSG. |
| getUnitInformation | dsg:STRING | A String representing the information for the unit with the corresponding DSG. |

### Alert and Warning Service

| Methods | Input | Response |
|---|---|---|
| register | area:STRING | A String for every indication, warning, or alert. |

## Adding New Services

To add a new service to the TITAN BCW you would create a new plug-in following the TITAN BCW architecture.

## Conclusion

Creating services with WSDL descriptions and SOAP or REST interfaces for the TITAN BCW is a fairly painless task. Since service creation can be accomplished merle by creating a JAVA class and generating a service description, extending the TITAN BCW in this manner is both simple and cost effective.

## Citation


1. AWS.docx https://svn.research.acincenter.org/wc2/code/OWL-S/services/aws
2. BBS.docx https://svn.research.acincenter.org/wc2/code/OWL-S/services/bbs
3. CODE https://svn.research.acincenter.org/wc2/code/OWL-S/services/code