# OADR2 VTN Reference Implementation – Jeff LaJoie

## Index

1. Project Overview
2. Design
3. Getting Started
4. Usage Overview
5. Current Features
6. Potential Improvements

## Project Overview

The project was done in the Java version of Play2 Framework, currently in version 2.0.4. The database system used is h2, and written simply to a directory and file created in the directory of the project. Utilizing Twitter Bootstrap CSS for design, as well as the Scala helper functions of Play2, the forms and display of pages are generated. Form binding to POJOs is handled via the framework based on the naming conventions of objects. JQuery is used for AJAX to refresh the pages in the Events display pages, as well as a Javascript date picker and time picker for the Event creation form. The library chosen for XMPP usage was Smack, and was tested with Openfire.

## Design

The reference implementation attempted to follow MVC principles, as the Play2 Framework encouraged this practice. Models consisted of classes, which were bound and created from the forms. Those forms in turn, were the views, and created using the Scala helper functions, generating .scala.html files. As such, the controllers did the routing of the scala.html files, as well as business logic for CRUD operations and rendering of web pages. As a rule, each model had a controller to manage it, and the associated views for such models; for an example, Events controller manages Event objects, and renders the pages of newEvent.scala.html, editEvent.scala.html and events.scala.html.

Singletons were used, both with Guice and lazy initialization. The module.Dependencies class is used by Play2 for configuration of the Guice plugin, and all singletons to be injected into the controllers package (note, controllers is the default injection package, and Play2 does not permit injection into multiple packages) are contained in the service package. The ProtocolRegistry is also a singleton, though not injected into controllers, and is used by the different protocols to determine which type of IProtocol the VEN contains, to be used for appropriate sending of payloads.

The protocol package is also a bit interesting and is used for VTN push scenarios. An interface, IProtocol, forms a contract that the abstract class, BaseProtocol, is required to fulfill. In turn, any additional protocol, other than XMPP and HTTP must extend this base class, as well as having a value added to the enum contained in the IProtocol. Within this package is also the ProtocolRegistry, that maps all VENs with a specific protocol to be used for send, based on the client URI. If the URI is not presently in the map, it is added to it to be more quickly at a later time.

For sending out the VTN, so as not to lock the application until all Events are done being sent, all events, and their associated target URI are added to a thread pool, PushService, which has all cores pre-started so as to immediately begin firing off the Runnable class, EventPushTask. Redis would be an equally acceptable, and potentially more scalable option in this situation, though for a simple reference there is little need.

## Getting Started

Begin by checking out the project from GitHub, link as of 15-11-2012

* git clone [git@github.com:EnerNOC/oadr2-vtn-refimpl.git](mailto:git@github.com:EnerNOC/oadr2-vtn-refimpl.git)

From there, Play2.0.4 needs to be installed. Instructions for doing so can be found at <http://www.playframework.org/documentation/2.0.4/Installing>.

Once the play command is on your path, you can start the application in either production or developer mode. The difference in developer mode is that any changes to code immediately affect the application upon reload or redirect.

Production:

* play start [-Dhttp.port=9000]

Developer:

* play run

When running for SSL and TLS, Play2 needs to be fronted with another web server, such as Apache2 or Nginx, and using reverse proxy to route the information

The web page should be accessible at <http://127.0.0.1:9000> by default.

## Usage Overview

As an example, I will walk through the usage of the OpenADR2.0aCertTest\_v1\_0\_4 testcases.vtn.pull.E3\_0430\_TH\_VEN test case.

1. Create a program with
   1. URI: <http://MarketContext1>
   2. Program Name: <http://MarketContext1>
2. Create a VEN with
   1. Program: <http://MarketContext1>
   2. VEN Name: TH\_VEN
   3. VEN ID: TH\_VEN
   4. Node Type: Pull
3. Run the E3\_0430\_TH\_VEN test case and follow the insutrctions
   1. Create an Event with
      1. Program: <http://MarketContext1>
      2. Event ID: test-event-one
      3. Priority: 0
      4. Intervals: 1
      5. Start date and time as the current time + 2 minutes
      6. End date and time as the current time + 7 minutes for an interval of 5 minutes
   2. Resume the test after the event has been created
4. You will be redirected to the page of all VEN responses, which will be refreshed with a VENStatus object displayed, containing the VEN\_ID, the OptType, and the Last Response time. To navigate to the general events page, select the Events tab at the top. To access this page once again, click on the entry of the EventID

## Current Features

* Program creation, retrieval, and deletion
* VEN creation, retrieval, and deletion
* Event creation, retrieval, update, and deletion
* AJAX update of both VENStatus and Event for EventStatus and OptType
* H2 database storage
* Form validation for creation of all objects
* Super intense Twitter Bootstrap CSS
* HTTP VTN pull response functionality
* Asynchronous XMPP VTN push functionality via thread pool

## Potential Improvements

* It is ugly, basic Twitter Bootstrap is unacceptable and should not exist on a professional web app. If it were modified with nicer colors (ex. Pantone EnerNOC) and formatting it would be a great addition.
* The Javascript Date and Time for Event creation is very odd, and a DateTime picker would be better. Not to mention the Time field currently goes from 11:45AM to 12:00AM, and noon is treated as 12:00AM rather than 12:00PM
* The selected Program of Events as well as the Date and Time are not pulled into the form when attempting to Edit that event
* Overly verbose queries, lack of joins for such queries and the use of the VENStatus could potentially be removed completely
* Play2 required some iffy type of implementation for injection and singletons, as well as requiring all handling of SSL and TLS from an external web service such as Nginx or Apache, which were supported in Play1
* Errors establishing the XMPP connection, on reload the connection is already established which causes a stream error, causing the connection to be dropped and no XMPP messages to be sent. This should never occur in production.