RoadWatch

What is RoadWatch?

RoadWatch is a tool for alerting road travellers to potential delays on a given Journey. It works by taking a journey and interrogating the UK Highways Agency RSS feed of road information for events whose locations overlap with the journey's route.

Design overview

RoadWatch has a core, presentation agnostic API whose function is to deliver road events to a client. Additionally RoadWatch has a Java Portlet based client that utilises Google Maps technology to render Road Events on a Google map instance running in a user's browser.

Constraints

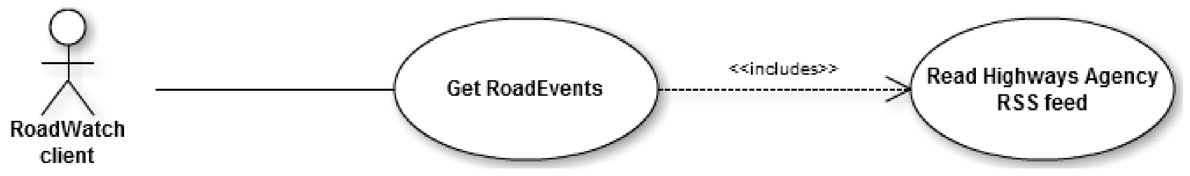
- System must be deployed within Liferay 6
- System must utilise Spring MVC for Portlet implementation

Principles

- Core functionality should be a self contained unit
- Core functionality should be unit tested
- Architectural layers should only depend on lower layers e.g. core should not depend on UI however UI may (and most likely will) depend on core.

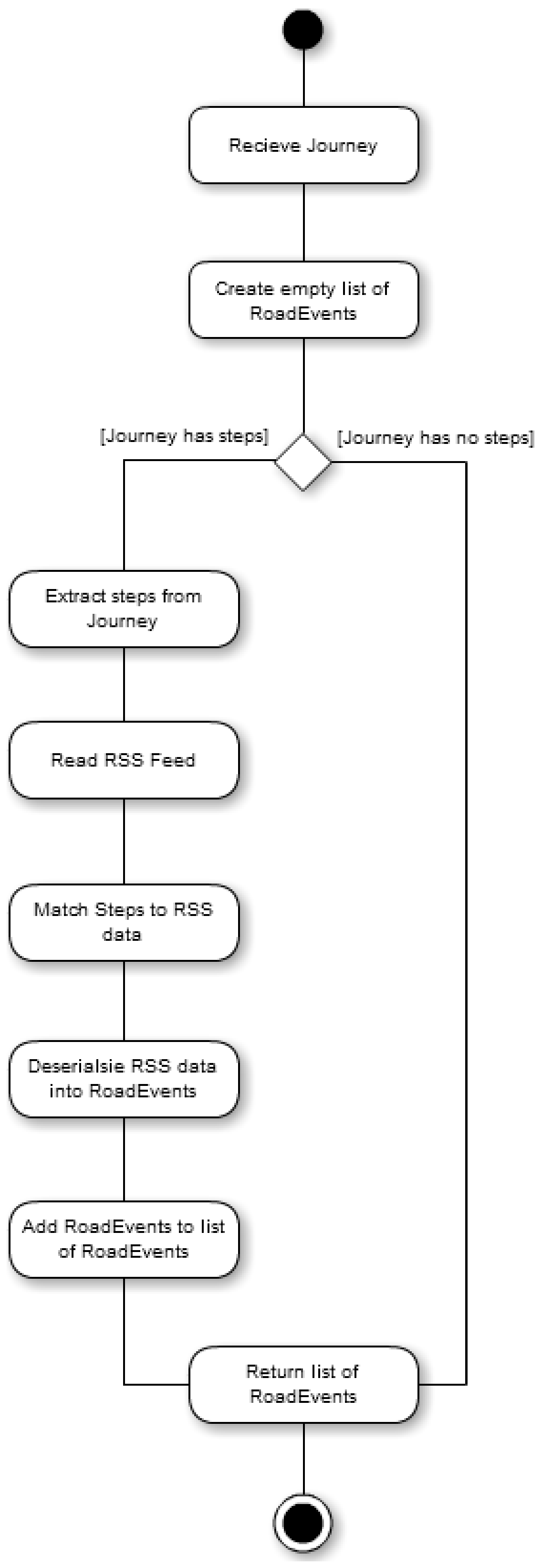
Functional view

This view is intended to describe the main use case supported by RoadWatch.



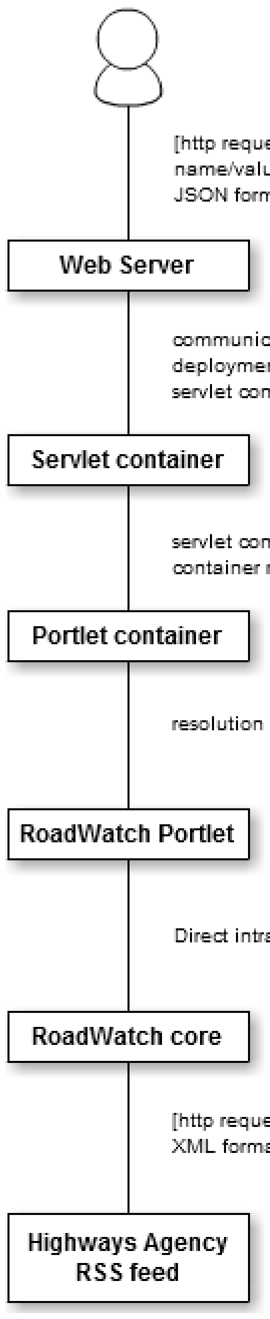
Process view

Describes how RoadWatch fulfils the main use case as defined in the Functional View section


Recieve Journey
Create empty list of RoadEvents
[Journey has steps]
[Journey has no steps]
Extract steps from Journey
Read RSS Feed
Match Steps to RSS data
Deserialsie RSS data into RoadEvents
Add RoadEvents to list of RoadEvents
Return list of RoadEvents

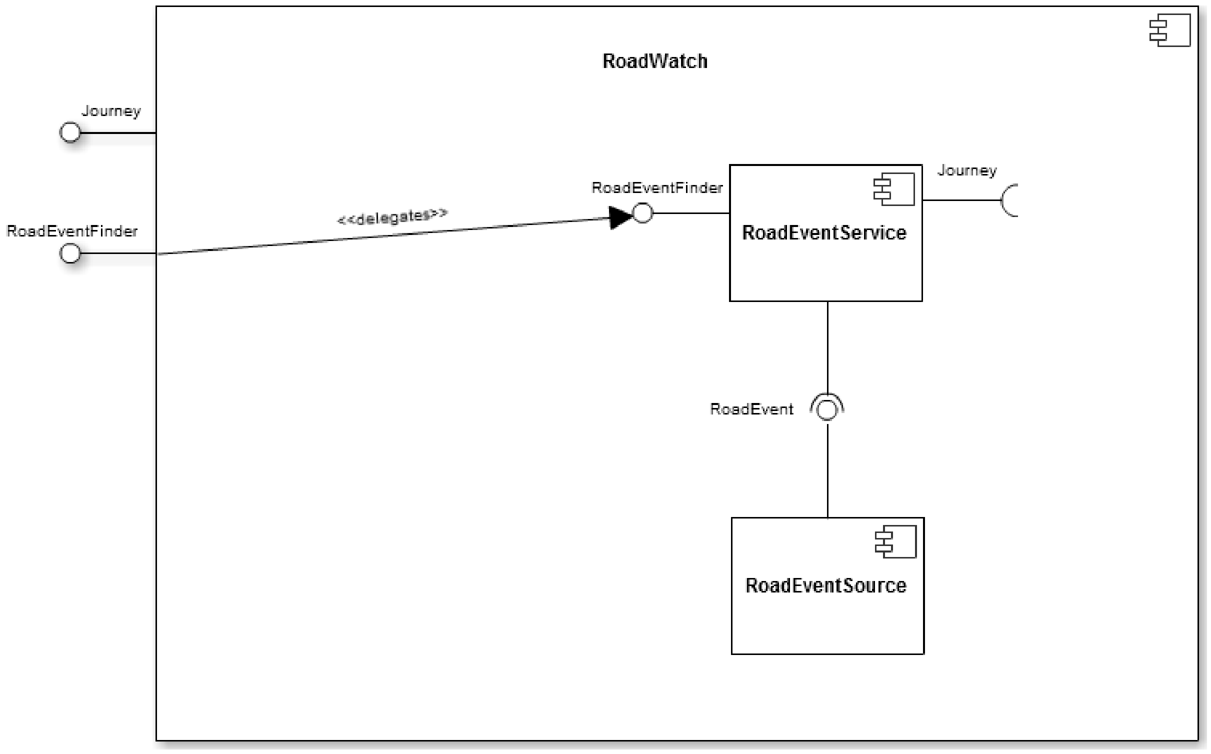
Logical view

Big picture view showing how the system is broadly structured and how each component of the system interacts


[http request/response]
name/value pair params for request
JSON formatted text is returned
Web Server
communication mechanism depends on deployment strucutre of web server and servlet container.
Servlet container
servlet container fowards requests to portlet container running in process
Portlet container
resolution is container dependent
RoadWatch Portlet
Direct intra-JVM communication
RoadWatch core
[http request/response]
XML formatted data is returned
Highways Agency RSS feed

Component view

This view shows the major component parts of the main RoadWatch component. Shows what interfaces they provide and consume.



Project structure on disk - *Noteworthy folders*

Note that the project can be imported directly into Eclipse.

- **css** - style sheets used in RoadWatch portlet. There is only 1 stylesheet used in the portlet (roadwatch.css)
- **images** - images used in RoadWatch portlet
- **js** - JavaScript libraries used in RoadWatch portlet. There are only 2 JavaScript libraries that the portlet's HTML presentation layer depends on. The first in the Google maps API which is hosted by Google. The second, roadwatch.js abstracts the majority of the logic required for the main road watch page to interact with the RoadWatch portlet running within the webserver. This roadwatch.js library file is the only file within the js directory.
- **lib** - compile-time dependencies
- **mocks** - classes used in testing. Provide mocked up data in a known state that can be used to validate assertions made by the unit tests.

- **src** - implementation code. Note that the portlet code is contained in this folder along with core RoadWatch code. At war build time these units are physically separated. The core API is jar'd up and placed in the WEB-INF/lib folder where it becomes a runtime dependency of the RoadWatch portlet code.
- **test** - JUnit tests of core RoadWatch code
- **WEB-INF**
    - **/jsp** - location of portlet's JSP files that are manipulated by the portlet controller contained in the /src folder. RoadWatch has a very simple UI supporting one use case therefore there is only one jsp file required.
    - **/lib** - portlet's runtime dependencies. This includes the RoadWatch jar that holds the core RoadWatch API that was separated out from the portlet code at war build time.

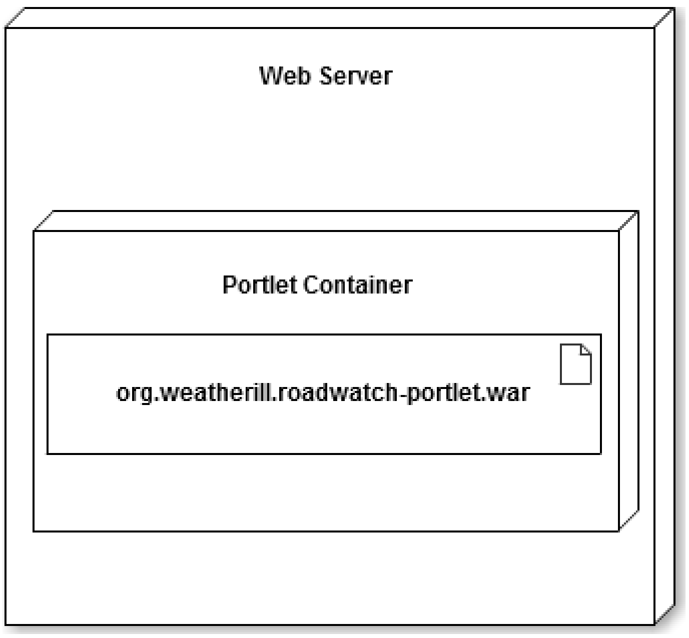
Building & Deployment

Building and deployment is done via an Ant build script (/build.xml). The default target ultimately produces a RoadWatch war file which then gets deployed into a Liferay installation. In order to do this build.xml makes use of a property in /build.properties that specifies the location of the liferay server. You will probably have to change this value to reflect the setup of your machine.

Note that it is assumed that you already have a Liferay installation up and running. If not (at the time of writing) Liferay can be downloaded along with the Tomcat web server from Liferay’s web site: www.liferay.com

Deployment view

The diagram shows where the RoadWatch war file that the build process produces is deployed to. Ultimately it will end up within the Portlet container which itself is hosted within a Web server. For brevity the servlet container layer that sits between the Web server and the portlet container has been omitted.



Testing

The RoadWatch portlet has a suite of JUnit tests to cover the core API

Additionally the portlet has been tested on the following browsers:

Google Chrome 19.0.1084.52 m

Internet Explorer 9.0.8112.16421