Developer Handbook- 2nd EvAAL competition

Evaluating AAL Systems through Competitive Benchmarking

http://evaal.aaloa.org

Special theme on Indoor Localization and Tracking

SmartHouse Living Lab

3-5 July, 2011

Madrid, ES

Developer Handbook:
2st EvAAL competitions

Contents

1 INTRODUCTION 4

1.1 ARCHITECTURE 4
1.2 COMPETITION PROCEDURE 5

2 REQUISITES 6

3 DEMONSTRATOR RUN 7

3.1 DOWNLOAD THE RUNNING ENVIRONMENT 7
3.2 IMPORT MAVEN PROJECT 7
3.3 RUNNING 9
3.4 STOPING THE APPLICATION 9

4 DEVELOPMENT PHASE 10

4.1 IMPLEMENTATION 10
4.1.1 Download example and template project 10
4.2 UPDATE CONFIGURATION 11
4.2.1 Pax run configuration 11
4.2.2 Running configuration 11

5 SOCKET PROTOCOL INTERFACE 13

5.1 ARCHITECTURE 13
5.2 RUNNING 13
5.3 SIMPLE SOCKET PROTOCOL 13
5.3.1 Sending localization events 14
5.3.2 Receiving device events 14
5.4 TWEAKING THE SERVER 15

6 DUMMY DEVICE TESTING 16

6.1 RUNNING 16

7 ADVANCED TOPICS 18

7.1 COMPILATION 18
7.2 ADVANCED PAX CONFIGURATION 19
7.3 QUICK RUN 19
7.4 IMPLEMENTING A UNIVERSAAL SOLUTION 19

Table of Figures

Figure 1 Competitor's Node Architecture 4
Figure 2 import Project 7
Figure 3 Maven proyect to import 8
Figure 4 Imported Project 8
Figure 5 Fast access to run command 9
Figure 6 Device Simulator GUI Interface 16

Figure 7 Click procedure to run maven install command 18

1 Introduction

This guide will show the different steps that should be followed to run the competitor application for the competition. First off, competitors will be guided through the setup to install the required software tools which are necessary for developing universAAL applications.

Once the setup is ready competitors will download and import the code. This will enable competitors to run a demonstrator.

Downloading a second piece of code, they will be able to start development by creating a class that implements certain java interface.

Finally the competitor will be guided to configure, run his/her application through the Pax Runner in order to be able to run it properly.

Alternately Competitors may interface with the evaluation nodes by means of a simple line command protocol over a TCP/IP socket.

1.1 Architecture

The code for the competition works over universAAL platform, which is eventually running over an Apache Felix OSGi container. To make things easier for competitors an adaptation layer has been added (displayed in blue in Figure 1).

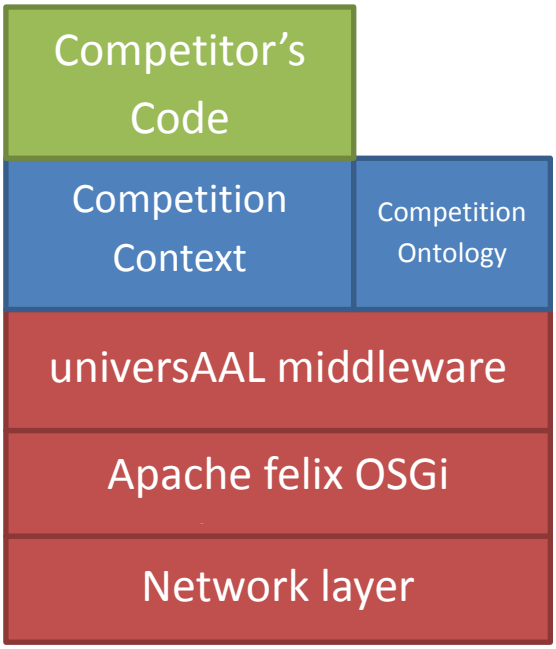

Figure 1 Competitor's Node Architecture

This guide will focus on how to interface with such layer, and how to run an instance in the competitor's node.

1.2 Competition procedure

First off, a platform running universAAL will be configured to listen to all events generated by the competitor, from the beginning to the end of the competition, as well as sending interesting context events to the competitor. This machine, connected to the (W)LAN, will be located in a computer room next to living lab and will be referred henceforward as evaluator's node.

The competitor should place his/her computer inside the living lab and plug it to the LAN, or connect to the WLAN. So all the events could be received and registered by the evaluator's node.

Once the competitor demonstrator is configured, the localization system will be evaluated in two phases as details "Annex 1 test procedure details" (read it for specific details):

Phase 1. In this phase each team must locate a person inside an Area of Interest (AoI). The AoI in a typically AAL scenario could be inside a specific room (bathroom, bedroom), in front of a kitchen etc. AOIs are defined here: http://evaal.aaloa.org/files/AOIs-coordinates.pdf

Phase 2. In this phase a person that moves inside the Living Lab must be located and tracked (we plan only 2D localization and tracking here). During this phase only the person to be localized will be inside the Living Lab. In this phase each localization system should produce localization data with a frequency of 1 new item of data every half a second (this will be also used to evaluate availability). The path followed by the person will be the same for each test, and it will not be disclosed to competitors before the application of the benchmarks. The environment will be made as much as possible similar to a house. This means that, if possible, there will typical appliances on, nearby wifi AP on, cellular phones on etc. In order to evaluate the accuracy of the competing artefacts, the organizers will compare the output of the artefacts with a reference localization system. Currently, the number and lengths of the paths used in the tests, and the number and position of AoIs. Details will be communicated to the competitors in advance.

Phase 3. This phase is organized as the former phase, but the competing artefacts will be evaluated in the presence of another person who moves inside the Living Lab. Only one person must be localized by the competing artefacts, the second one will follow a predefined path unknown to the competitors.

2 Requisites

The first step is to install development environment:

1. JDK[1]
2. Eclipse 3.5.2[2] ("Galileo SR2") or higher, as long as the M2Eclipse[3] pluging is able to be installed. Eclipse 3.7.x [4]has this plugin installed by default, therefore it is recommended to use the latest version of eclipse.
3. Together with a SVN client (your preferred choice, for example Tortoise[5] or any eclipse SVN plugin).

[1] http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u25-download-346242.html
[2] http://archive.eclipse.org/eclipse/downloads/drops/R-3.5.2-201002111343/index.php
[3] http://eclipse.org/m2e/
[4] http://www.eclipse.org/downloads/packages/eclipse-classic-372/indigosr2
[5] http://tortoisesvn.tigris.org/

3 Demonstrator Run

3.1 Download the running environment

To download the necessary files to run an example (and later the competitors implementation), use the SVN client to "check out" the following URL:
https://svn.aaloa.org/projects/evaal/Editions/Ed2012/LocationTrack/SDK-ev12/

Create your working copy (local location) in any location like c:\dev\svn\evaal\, it is also recommended to download it directly into the eclipse workspace.

3.2 Import Maven project

Once the code is downloaded, to import the code open Eclipse and follow these steps:

1. Go to "File -> Import" (or left click on the package explorer pane and then select "Import"). The following dialog will appear:

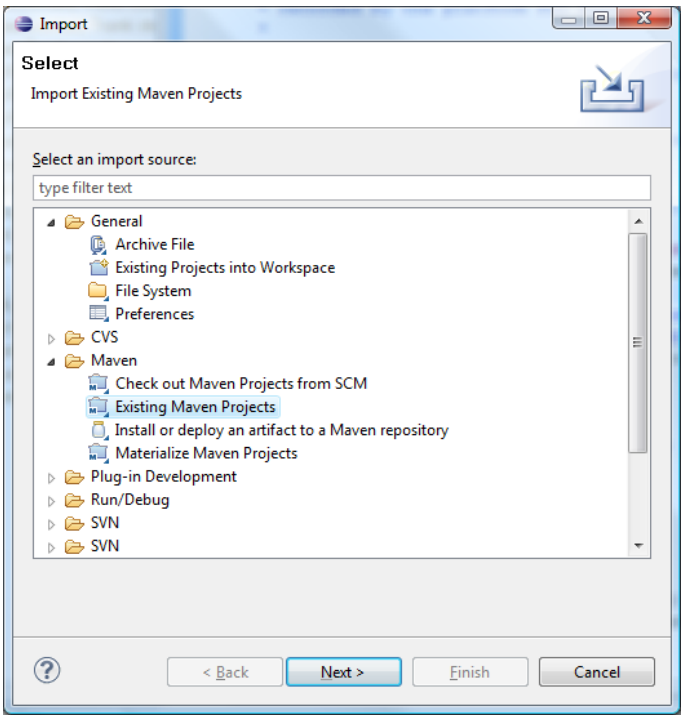

Figure 2 import Project

2. Select Existing Project to import:

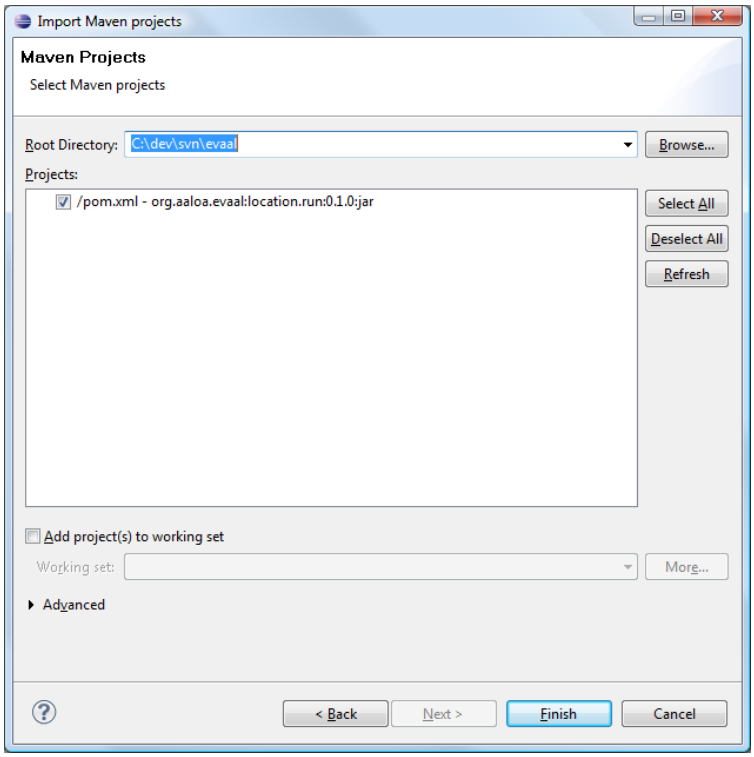

Figure 3 Maven proyect to import

3. Click finish, you should have now a new project in your package explorer:

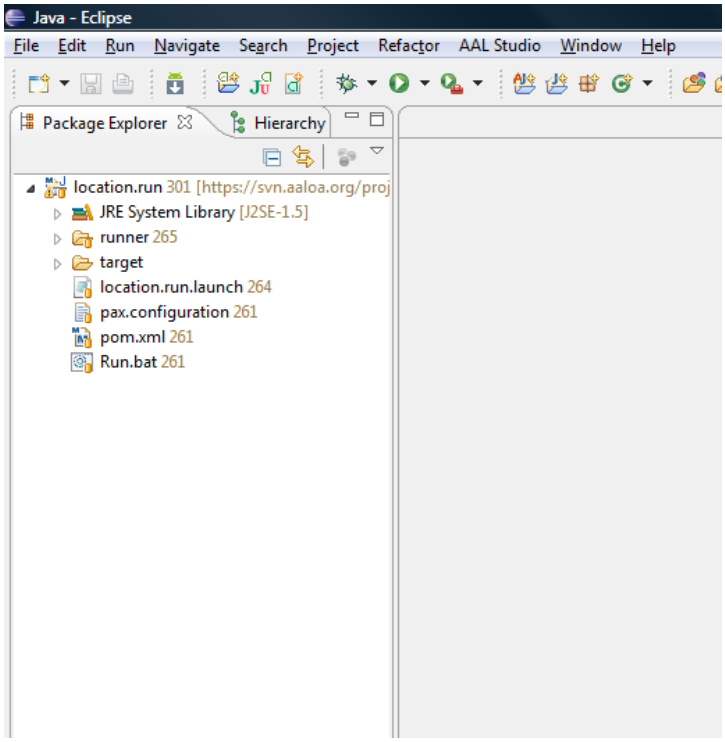

Figure 4 Imported Project

3.3 Running

To run just left click over the location.run project and select “Run as -> "Run as -> Maven build" (see Figure 5).

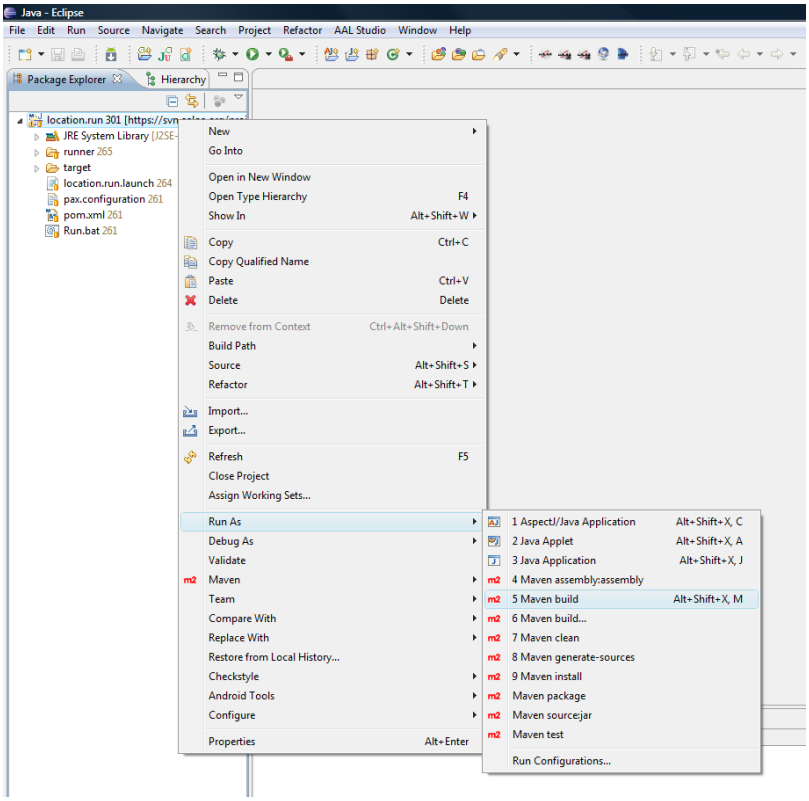

Figure 5 Fast access to run command

This will download all necessary bundles and libraries (check the console), so make sure there is internet connection. This may take some time, specially the first time, so please be patient.

[NOTE] If you use “Maven build…” (emphasis on the dots), this will generate a new running configuration. It is recommended not to use this option, and if clicked accidentally remove the configuration in the “Run -> Run Configurations…” menu by right clicking on the “Maven Build-> location.run(1)” element.
[NOTE] If this does not start any running configuration just add the appropriate run configuration. Using the “Maven build…” (note the dots), on the right click menu over location.run, the new run configuration should have “pax:run” as goal.

3.4 Stoping the application

Once the test is finished, stop the run type *“stop 0”* in the console.

Clicking on the red square button ( ), will leave the application running in background, and cause that subsequent runs will use out-dated classes. So **DO NOT** use this button ( ).

4 Development Phase

4.1 Implementation

An interface is provided for the competitor to implement. This interface (`org.aaloa.evaal.location.competition.Activator.CompetitorInterface`) provides 4 methods to be completed:

- *getCompetitorId()* : must return an identification string for the competitor team.
- *start()* : is the starting point for the competitors code.
- *stop()* : instruct when to stop the competitors code.
- *Handle(CompetitionDeviceView)* : this method will be called when a competition device is (de)activated. To access interesting information such as the coordinates of the devie use the methods in the interface: *org.universaal.ontology.competition.owl.CompetitionDeviceView*

When the competitor wises to publish a location coordinate, he/she should use `org.aaloa.evaal.location.competition.Activator.Activator.locate(double, double, long, int[])` method.

Any competitor's code must comply with the following:

- Built into one or more .jar libraries/bundles (see 4.2.2 to include them in the running configuration)
- One class should implement this interface: `org.aaloa.evaal.location.competition.Activator.CompetitorInterface.`
  (remember to configure the run environment, see section 4.2.2, so it is able to locate this class)

4.1.1 Download example and template project

For your convenience there is a template maven project. Download and import (just like in sections 3.1 and 3.2) the following project:

https://svn.aaloa.org/projects/evaal/Editions/Ed2012/LocationTrack/SDK-ev12/location.sample/

There you should find a Dummy class and a Template class; both will comply with the implementation requisites for competitor's code. In fact Dummy class is the class executing during the demo run (see section 3), so this should be a good example to understand the development phase.

The template class may also be used to start new development; it provides the skeleton that should be completed. It is recommended to copy the template into a new class, and then develop on top of the new class.

This maven project is ready to be built into the application so it is recommended to develop using this project, see section 7.1 on how to compile it. After compilation the run procedure (see sections 3.3 and 3.4) will import this project by default, so the configuration steps are minimal (just the procedure explained in 4.2.2, if there are no extra libraries to be loaded).

4.2 Update configuration

Before running the developed solution some configuration of the running environment is needed, in this section the configuration needed is explained. Once configured you can run your code just follow section 3.3; and section 3.4 to stop.

4.2.1 Pax run configuration

Edit the file /location.run/pom.xml to import the necessary jars and bundles into the run. This pom file is configured to use the pax provisioner plugin [6]for maven.

The pom.xml file should include a *provision* tag for each jar or bundle needed in the run in the maven-pax-plugin configuration section. Each tag should have the form <provision>url</provision>, where the url depends on whether the imported jar is a bundle or not, and whether it is listed in the maven repository or it is stored locally in a file use the following table to include the correct libraries:

	Library	Bundle
Maven	*wrap:mvn:group.id/artifact.id/version*	*mvn:group.id/artifact.id/version*
File	*wrap:file:/C:/some/directory/file.jar*	*file:/C:/some/directory/file.jar*

These urls are provision commands[7] used by Pax Runner.

If the location.sample project is used (instead of a new one), and no other jar or bundle are necessary then this configuration should be ready to go, as it already imports the generated jar. Although a compilation (see section 7.1) should be done each time the code changes, in other that the latest version is the one actually running.

There are already provision commands, which should not be removed:

- Provision of universAAL middleware
- Provision of Location framework that includes the Interface and adaptation code.
- Provision of a Logger bundle, this bundle will log both your location publications and device events, this log is stored in /location.run/runner/log and you will be asked to send it to the committee after the competition.
- Provision of a socket client, this bundle is not to be started unless there is a failure of universAAL middleware automatic peer discovering, in such unlikely event you will be instructed insitu on how to proceed.

4.2.2 Running configuration

By default the project runs a default example code (Dummy) that publishes random locations.

[6] http://www.ops4j.org/projects/pax/construct/maven-pax-plugin/provision-mojo.html
[7] http://ops4j1.jira.com/wiki/display/paxrunner/Provisioning

To change this behavior, configure the *CompetitorClass* property to point your own implementation of *CompetitorInterface* in the file: /location.run/runner/configurations/location.competition/competition.properties.

This is a required step to make sure your code is correctly run.

5 Socket protocol interface

Due to integration difficulties it may arise with the interface provided, the EvAAL board has decided to offer a simpler interface with universAAL. This interface is through a socket, a server is run in the evaluator's node and the competitor just has to inform the server through sockets. This section describes this solution.

5.1 Architecture

The architecture of this solution is based on another concept of adapter, instead of receiving Java calls it receives commands through an IP socket. Then it interprets these commands and transforms them into universAAL context events. This process is inverted to produce the device notifications.

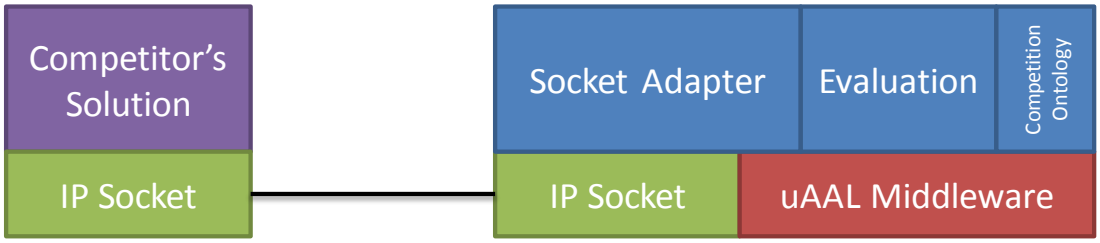

Figure architecture of the Socket Adaptor solution

5.2 Running

To run an instance, for testing purposes, follow these simple steps:

1. Download file from: https://svn.aaloa.org/projects/evaal/Editions/Ed2012/LocationTrack/SDK-ev12/
2. Un-zip the file in any location of your convenience.
3. Double click on the *run.bat* script file (note this script can also be run from a console, and will work also in Unix-based OS)

In windows, a terminal should appear, displaying info and debug messages. To ensure that the SocketAdapter has correctly started look, within the few last lines, for something like the following:

```
14:23:40.065 - INFO  - [Thread-2]
org.aaloa.evaal.location.socket.competition.Activator Opening listening
socket at 4444
```

There will be an instance running in the living lab as backup for those competitors who opt for this solution, so it is not needed to run an instance in the competitor's node during the competition. The IP of the evaluator's node will be revealed at the competition.

5.3 Simple Socket Protocol

Once the socket is opened, the communication will be trough textual mode. The default port is 4444 (over TCP).

5.3.1 Sending localization events

The parameters should be entered, separated by single white space, in the following order:

```
<ID> <X> <Y> <T> {<AOI>}*
```

Where:

- **<ID>** is a string identifying the competitor issuing the location point.
- **<X>** is the X coordinate of the location point, represented in decimal form (valid examples: “0 “1” “1.2” “1.269856”; all without quotation marks). Remember the expected resolution is that of a double (IEEE 754 binary64 standard) higher resolutions would not be parsed. The reference coordinates can be downloaded here: http://evaal.aaloa.org/images/LL-coordinates.jpg.
- **<Y>** same as <X> but for the Y coordinate of the location point.
- **<T>** is an integer representing the time at which the person was located in the current location point described. This is intended to represent time since epoch in miliseconds (otherwise known as POSIX time), so this is any integer from 0 to 4294967295. Although not recommended this parameter can be also expressed in relative time.
- **<AOI>** is a number representing the area of interest at which the person is. The AoIs’ identifiers will be supplied when in the laboratory, there can be none or n areas of interest; all of them should be integers and separated by a space.

Each command should end with an “end of line” character. The ScoketAdapter will respond a string “Published” if it has been parsed correctly and the event has been published or “NotParsed” if the command was not correctly formatted; both followed by an “end of line” charracter.

These commands can be tested by means of any telnet client, connecting to wherever the SocketAdapter is running (normally in localhost, the same computer as the client is running) on port 4444. Connect with the server and input textually the commands.

```
CompetitorD 0 0.235 1309782587440
Comp_D 1.569889 2.45694 1309782569435 1 2 3
D_comp 5.697946 3 13097825693165 4
D 0 0 0 0
D 0.5698989875687987 0.6859865 15 10 11 20 21 22 40 41 43
```

Table 1 Examples of valid commands

5.3.2 Receiving device events

Over the same port communication you will receive the events, the commands for such events have the following structure

```
Device: <ID> <T> <ACTIVE> <X> <Y>
```

- **<ID>** is a string unique per device.
- **<T>** is a long integer representing the time, in milliseconds since epoch, at which the device had changed state.
- **<ACTIVE>** is “true” or “false” indicating the status of the device.
- **<X>** is the X coordinate of the location of the device, represented in decimal form (valid examples: “0 “1” “1.2” “1.269856”; all without quotation marks). The reference coordinates are visible here: http://evaal.aaloa.org/images/LL-coordinates.jpg
- **<Y>** same as <X> but for the Y coordinate of the location of the device.

These commands are sent asynchronously, so one of these commands can be sent before a publication confirmation.

5.4 Tweaking the server

The SocketAdapter server can be configured through the file located at: *configurations\location.socket.competition\competition.properties*

In this file the Port, delimiter used to separate parameters, maximum number of connections, and the type of new line after the respond string, as well as these status strings is configurable.

Parameter	Description	Default
port	Port to listen for connection requests	4444
delimiter	Delimiter for parameters in simple socket protocol	<White Space>
max_connections	Maximum number of simultaneous connections	10
newLineReturnWords	Type of new line after return strings, “\n” is a new line and “\r” is a carriage return.	\n\r
correct	The string outputted when the command has been parsed and published	Published
incorrect	The string outputted when the command has not parsed.	NotParsed

The evaluator’s node will be configured with the default values.

6 Dummy device testing

This edition of EvAAL location track includes a new challenge; competitors are able to user contextual information to refine their localization events. Since the devices used in the Living Lab cannot be shipped around to competitors to make their tests, a Device simulator is provided.
This application will enable competitors the tests of their reasoning/refining algorithms, by offering a simple way to inject device events into the interfaces described above. This application will worth with both interfaces, just by running it in the same network. There may be as any number instances.
It is recommended that the test procedure, done in the competitor's lab to validate this feature, includes several PCs. Being their position know, and by pre-filling the form, actuation events can be publish by simply clicking the "Publish" button when the actor is near.

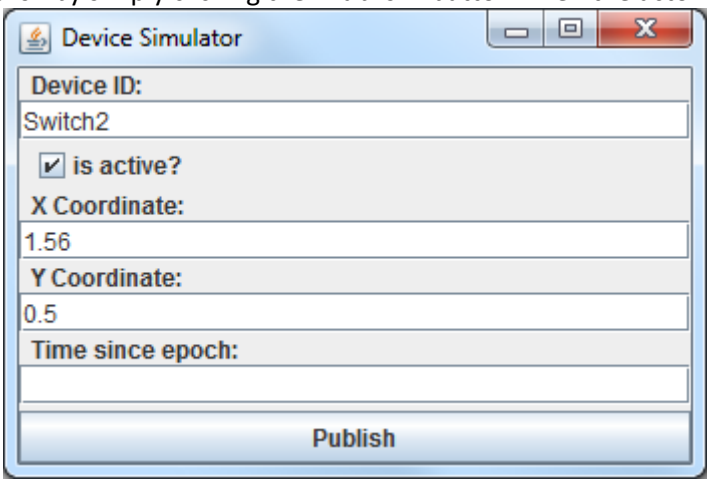

Figure 6 Device Simulator GUI Interface

The form is straight forward; it has 5 input parameters and a button:

- Device ID: a unique identification string per device.
- Is Active: the status of the device, if for example the simulated device is a light switch, then it can be either turned off (inactive) or on (active).
- X and Y coodinates: the coordinates from a reference point.
- Time since epoch: the time at which the device is actioned. If left blank it will automatically take the time epoch when the button is pushed.
- A Device event will be published each time the publish button is pressed, so this should be reflected in the competitor's code regardless of the interface used.

6.1 Running

To run an instance, for testing purposes, follow these simple steps:

1. Download file from:
   https://svn.aaloa.org/projects/evaal/Editions/Ed2012/LocationTrack/SDK-ev12/

2. Un-zip the file in any location of your convenience.
3. Double click on the *run.bat* script file (note this script can also be run from a console, and will work also in Unix-based OS and MacOS)

In windows, a terminal should appear, displaying info and debug messages, along with the GUI interface. To close it simply click on the “x” on the dialog.

7 Advanced Topics

This section should only be read by experienced developers, or to find solutions that may be compatible with your implementation.

7.1 Compilation

It is highly recommended to use maven project management, as all projects supplied are maven projects. If you don't know how to use maven please refer to the maven project home page for more information:

http://maven.apache.org/

If you just need to compile a maven project, then maven script install will do it. Once you can see your project in the explorer view press right click and "Run As -> Maven install"

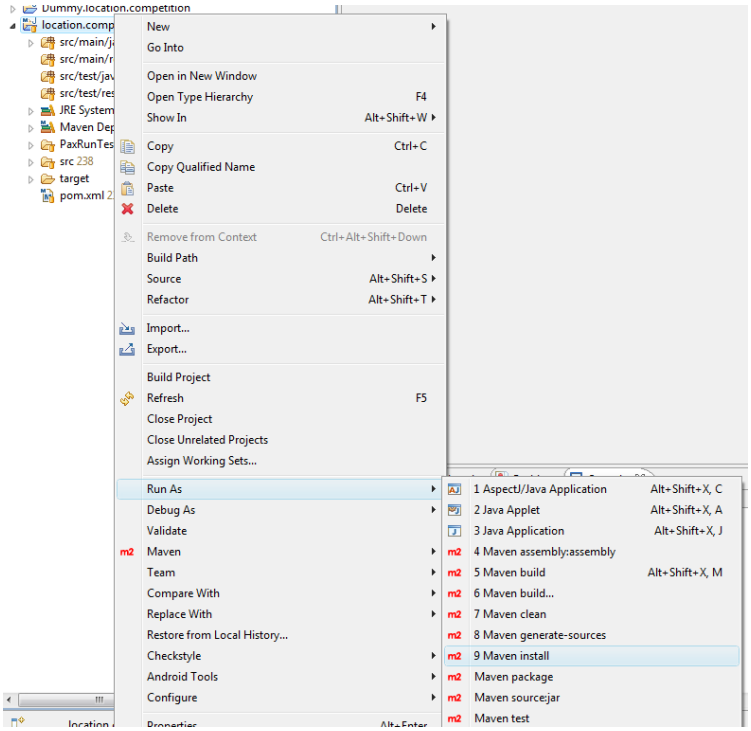

Figure 7 Click procedure to run maven install command

All required platform artifacts should be downloaded (check it on the eclipse console). And the project will be compiled. This step may take a few minutes the first time, please be patient.

[NOTE]
If you get the following error:

…
[INFO] Compiling 5 source files to C:\test\location.competition\target\classes
[INFO] --
[ERROR] COMPILATION ERROR :
[INFO] --
[ERROR] Unable to locate the Javac Compiler in:
C:\Program Files\Java\jre6\..\lib\tools.jar

Please ensure you are using JDK 1.4 or above and
not a JRE (the com.sun.tools.javac.Main class is required).
In most cases you can change the location of your Java
installation by setting the JAVA_HOME environment variable.
…
Go to "window -> preferences -> java -> installed JRE's" menu, and remove all except the latest JDK.
If there is not any JDK, add one. This is done by clicking on the "Add" button, select "Standard VM", click "Next", select "JRE Home" as your JDK local intallation folder (it should be some thing like: C:\Program Files\Java\jdk1.6) and click "Finish"

7.2 Advanced PAX configuration

Each artifact may have extra configuration that are appended after the url, one or more configurations may be appended:

- **@<run_level>**: <run_level> is an integer representing the start level of the bundle. If present and OSGi Start Level Service is available then the bundle is installed with that specific start level.
- **@nostart** : present the bundle should not be started. Default, if the value is not present the bundle will be started depending on the service configuration. This configuration can be useful if you do not wish to start the location.competition bundle (an there for your code) right away
- **@update**: if present and the bundle was already installed before, the bundle will be updated. The default behavior, if the value is not present, the bundle will be updated or not depending on the service configuration. Useful for those bundles/libraries which will change during development and testing.

These configurations are especially useful to make sure the bundles are updated to their latest version and that they start in the desirable order.

7.3 Quick run

The running procedure, explained in section 3.3 may take a lot of time each time it is executed. This is because it updates all the bundles, libraries and necessary configuration. But once this is done the environment can be very quickly executed by running the script:
location.run/runner/run.bat

Remember this run method will **not update** your bundles or libraries; so you may have to either run as explained in section 3.3 or update manually the jar files. That is why this method is discouraged

To manually update a Bundle just copy it, replacing the old one, in the folder:
location.run/runner/bundles

7.4 Implementing a universAAL solution

If you are interested in the development over universAAL, or if you already know how to develop over universAAL platform, there is also the possibility of creating your own universAAL context publisher and discard de adaptor. Then you may use the adaptor project as template:

https://svn.aaloa.org/projects/evaal/Editions/Ed2012/LocationTrack/SDK-ev12/location.sample/

Remember to run only one context publisher, so delete the location.competition bundle from the pax configuration (see section 4.2.1).

The format of the context events should have a `Competitor` (all these classes are inside `org.universaal.ontology.competition.owl` Package) as subject, `Competitor.PROP_PUBLISHES` as predicate. Remember that this property is a *LocationPoint* so you will need to update this property before publishing each event.

The format of the events received from devices are:

- Subject: *CompetitionDevice*
- Predicate: *CompetitionDevice.PROP_PHYSICAL_LOCATION*
- Object Type: *org.universAAL.ontology.location.position.Point*

For more development details on how to develop for universAAL please refer to: http://depot.universaal.org