Pushing dynamic and ubiquitous interaction between services **L**everaged in the Future Internet by **A**ppl**Y**ing complex event processing

Project number:	258659		
Instrument:	Collaborative Project		
Thematic Priority:	ICT-2009.1.2: Internet of Services, Software and Virtualisation		
Start Date:	01/10/2010	**Duration:**	36 months

D5.1.2 – Integrated PLAY platform & platform Manual V1

WP5	Platform and Integration

Due date:	M20
Submission Date:	19/06/2012
Organization Responsible for the Deliverable:	EBM
Version:	V1.0
Status:	Ready for submission
Abstract:	Description of the implementation and integration undertaken, detailed definition of prototypical system and manual

Author(s):	Christophe Hamerling Laurent Pellegrino Roland Stühmer Philippe Gibert Yiannis Verginadis	EBM INRIA FZI ORANGE ICCS
Reviewer(s):	Philippe Gibert Louis Plissonneau	ORANGE

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission)	

* - put **X** in proper field

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	25/04/2012	ToC	CIM
0.2	11/05/2012	SAN/ESR section	ICCS
0.3	11/05/2012	EventCloud section	INRIA
0.4	14/05/2012	DSB section	EBM
0.5	15/05/2012	Governance and Monitoring sections	EBM
0.6	29/05/2012	Add DCEP section. Add Platform usage section. Update Applications. Add Executive Summary and Introduction Add Event Adapter section	FZI, ORANGE, INRIA, EBM
0.7	30/05/2012	Internal Review	ORANGE
1.0	31/05/2012	Final version	EBM

Table of Contents

1.1 List of tables

1.2 List of figures

1.3 List of listings

Executive Summary

The main goal of the PLAY project is to develop and validate an elastic and reliable architecture for dynamic and complex, event-driven interaction in large highly distributed and heterogeneous service systems. The PLAY platform is composed of several components also called modules. Those components are linked together using the platform event approach. The goal of the current deliverable is to describe how to get, install, configure and run those components. It also focuses on how the PLAY platform can be used by illustrating this usage with real software developed within the project.

2 Introduction

The current deliverable provides all the technical information for setting up the PLAY platform. The PLAY platform is composed of several independent software components linked together using integration techniques described on previous WP5 deliverables.

The document gives details on how to build the modules from their sources, how to configure and start them. It also provides several information and recommendations such as hardware requirements and software dependencies (Operating System, Virtual Machines, Memory, Hard disk space …).

The last sections are usage-oriented and describe how the whole PLAY platform and its components are used. It gives details on how events coming from several Internet sources are injected in the platform, how these events are used on some of the PLAY applications we develop and how the events are used inside the platform.

2.1 List of Acronyms

Acronym	Definition
DCEP	Distributed Complex Event Processing
DSB	Distributed Service Bus
ELA	Event Level Agreement
ESR	Events Subscriptions Recommender
JAR	Java Archive
JVM	Java Virtual Machine
OS	Operating System
RDF	Resource Description Format
RDFS	Resource Description Format Schema
SAN	Situation Action Network
SDK	Software Development Kit
WSN	Web Service Notification

3 Platform Components

The PLAY modules can be downloaded independently as described in the next subsections or from the following HTTP locations:

- **Distributed Service Bus**
  - Binary: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/dsb-distribution.zip
  - Sources: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/dsb-distribution-src.zip
- **EventCloud**
  - Binary: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/event-cloud-bundle.zip
  - Sources: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/event-cloud-bundle-src.zip
- **Distributed Complex Event Processor**
  - Binary: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/dcep-distribution.jar
  - Sources: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/dcep-distribution-src.jar
- **Governance**
  - Binary: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/easiergov-installer.jar
  - Sources: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/easiergov-src.jar
- **Monitoring**
  - Binary: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/easierbsm-installer.jar
  - Sources: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/easierbsm-src.jar
- **ESR**
  - Binary: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/esr-v1.0-bin.zip
  - Sources: http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/esr-v1.0-src.zip

A complete archive containing all these files is available at http://www.fzi.de/downloads/ipe/play_fp7/m20/d521/play-modules.zip.

3.1 Event Cloud

To run EventCloud instances, Java 6 or greater is required. If you are interested to get EventCloud binaries you can either follow the instructions given in the *Generating binaries and bundle* section to build them from sources, or you can download them directly through Maven repositories. Releases are put into Maven Central (which is the

default Maven repository accessible without configuring anything) and snapshots (versions under development) are regularly put into the OSS repository[1].

3.1.1 Generating binaries and bundle

Thanks to Apache Maven, an archive containing all necessary binaries, dependencies and scripts to create and deploy EventCloud services can easily be generated from the sources. To build this bundle, Maven 3.0.3 or greater is required. In case you do not want to do it manually, nightly builds and stable versions are available from http://eventcloud.inria.fr/binaries/.

The process to generate the EventCloud distribution is the following: First, you have to download EventCloud source code. The sources are publicly available and can be browsed or downloaded from http://eventcloud.inria.fr/.

Once you have retrieved the sources you have to move to the folder where you have checked out the project. Now, you are at the root of the EventCloud hierarchy which is composed of several Maven modules and sub-modules. Please execute the command `mvn clean install -DskipTests` to package and install each JAR into your local maven repository. Then, you are ready to federate all these JARs into a bundle and to populate it with some scripts that ease the deployment of services. To make it, move inside the folder `event-cloud/event-cloud-bundle` and run the command `mvn assembly:single` to find a bundle named `event-cloud-bundle-x.y.z.tar.gz` in the `target/` directory once the execution is terminated.

3.1.2 Install

The installation of an EventCloud distribution consists only in unzipping the `event-cloud-bundle` archive where you desire. In the next section we will refer to `%EC_BUNDLE_HOME%` as the absolute path to the directory where you have unzipped an EventCloud bundle.

3.1.3 Configure

Regarding the configuration we distinguish two parts. The first one is about settings that are mandatory to deploy and to communicate with EventCloud services. The second explains how to configure the different properties that may be used by the EventCloud services when you start them.

3.1.3.1 Filtering Configuration

To communicate with the services that are deployed thanks to a script provided with an Eventcloud bundle (as described in the *Run* section), several ports have to be unfiltered:

1 In output all the ports must be unfiltered.
2 In input the following range of ports must be unfiltered:

Port range	Comment
8080-8089	Ports that may be used to communicate with EventCloudsRegistry(s) and EventCloudManagement webservices
1100-1299	Ports that may be used to communicate with EventClouds

[1] http://nexus.sonatype.org/oss-repository-hosting.html

9000-9999	Ports that may be used to communicate with proxies

Table 1 - Event Cloud ports configuration

In case you are using proxies to communicate with an EventCloud, the JVM in which the proxies are created has to be launched with the following arguments:

- `-Dproactive.communication.protocol=pnp`
- `-Dproactive.pnp.port=X` where X is a number between [9000, 9999]
- `-Dproactive.http.port=Y` where Y is a number between [9000, 9999] and is different of X
- `-Djava.policy=/path/to/java.policy` where java.policy contains:

```
grant {
  permission java.security.AllPermission;
};
```

If the machine on which on you are running the application is behind a NAT with a private address, you have to use the following property to specify a reachable public address (after having redirected the necessary ports to the private address) :

- `-Dproactive.hostname=xxx.xxx.xxx.xxx`

3.1.3.2 Services Configuration

EventCloud services can be customized through properties. To specify a property to use you have to update the configuration file in `%EC_BUNDLE_HOME%/resources/eventcloud.properties`. A list of all the properties whose the values may be altered is available from http://eventcloud.inria.fr/javadoc/fr/inria/eventcloud/configuration/EventCloudProperties.html

3.1.4 Run

An Eventcloud distribution must be started by executing the script `infrastructure-launcher.sh` provided in `%EC_BUNDLE_HOME%/scripts/`. This script is currently UNIX dependent and makes use of the *inotify-tools* library. Please check that this library is installed on your system before to run any script (e.g., to install it on Fedora use the command `yum install inotify-tools`).

The range of ports defined to deploy EventCloud services may be changed by editing the variables declared in the infrastructure-launcher script. In that case, do not forget to update your unfiltering configuration according to your modifications.

Running the script named `infrastructure-launcher.sh` will deploy an EventCloudsRegistry (which is in charge to keep in mind the EventClouds which are deployed) and an EventCloudManagementWebservice whose endpoint is printed on the standard output. The role of this webservice is to manage (list, create, deploy and destroy) eventclouds and proxies. To interact with it, entities have to invoke methods on this webservice or to use the eventcloud-manager web application we have developed.

The eventcloud-manager is a simple web application that may be used to easily manage eventclouds and proxies for a given eventcloud management webservice through a webpage. The source code is open source and hosted at Bitbucket. Requirements and configuration information are available on the project website:

https://bitbucket.org/lp/eventclouds-manager/

Eventually, you can stop the EventCloud services which have been deployed with the `infrastructure-launcher.sh` script by using the `-k` argument:

```
%EC_BUNDLE_HOME%/scripts/infrastructure-launcher.sh -k
```

3.2 Distributed Complex Event Processing

The Distributed Complex Event Processing component (DCEP) is implemented in Java and Prolog: Prolog is used for an efficient, rule-based inference engine to infer new events at the core of the engine whereas Java is used to hold everything together, enable distributed communication and provide usable APIs using service interfaces and component-oriented remote invocations. Prolog and Java "speak" to each other using JPL[2].

Prerequisite for running DCEP is a working and installed PLAY Event Cloud which can be contacted via the Internet.

DCEP is multi-platform. However, it is important to note that Java and Prolog must either both be 32 bit or both be 64 bit, otherwise JPL cannot work to connect them.

DCEP has been tested on:

- Windows 7 32bit
- Windows 7 x64
- Linux CentOS 5.8 x86_64

3.2.1 Obtain the Source Code

Using Apache Maven an archive containing all necessary binaries, dependencies and scripts to deploy DCEP services and platform services can easily be generated from the sources. To build this bundle, Maven 3.0.3 or greater is required.

1. Install Maven and Subversion (e.g. so that the commands "`mvn`" and "`svn`" are available on the command line). Alternatively this can be done using Eclipse plugins of your choice.
2. Create a new directory and "cd" to it, (e.g. "`cd c:\play-platform\`")
3. Get the sources from SVN (e.g. "`svn checkout https://svn.petalslink.org/svnroot/trunk/research/projects/play`"). You might need to inquire for permissions to access the repository first. Read access is available using the anonymous/anonymous credentials.

3.2.2 Compile

1. Enter the root folder of the sources (e.g. "`cd play`")
2. Compile the platform (e.g. "`mvn install`" or "`mvn install -DskipTests`". The latter is faster and during the prototype phase might circumvent some errors.)

3.2.3 Install

1. Install a Java JRE or JVM (e.g. Java SDK 6 Update 31 64-bit for Windows)
2. Install SWI-Prolog (e.g. SWI-Prolog 5.10.2 for Windows XP/Vista/7 64-bit edition)

[2] JPL, A Java Interface to Prolog: http://www.swi-prolog.org/packages/jpl/java_api/index.html

3.2.4 Configure

There is nothing to do here, currently. The command line parameters in the next section will suffice.

3.2.5 Run

1. Go to the subdirectory of play-dcep-distribution (e.g. "`cd play-dcep\play-dcep-distribution`")
2. Start DCEP (e.g. "`java -Dproactive.communication.protocol=pnp -Dproactive.pnp.port=9003 -Dproactive.http.port=9004 -jar dcep-distribution.jar`")

DCEP will connect to the Event Cloud to get simple events and DCEP will open the Query Dispatch Service to wait for new event patterns.

3.3 Distributed Service Bus

The Distributed Service Bus (DSB) runs with the help of a Java Virtual Machine version 6 or later. The binary can be downloaded from http://research.petalslink.org/display/petalsdsb/Download or generated following the instructions given in the following section.

Note that the DSB may need at least 1024 Mb of memory to run and over 200 Mb of hard disk space to extract all required artifacts.

3.3.1 Build binary from sources

Sources of the DSB can be retrieved from its github repository at https://github.com/PetalsLinkLabs/petals-dsb by downloading a snapshot of the repository at https://github.com/PetalsLinkLabs/petals-dsb/zipball/master or by cloning the repository using git:

```
git clone git://github.com/PetalsLinkLabs/petals-dsb.git
```

The DSB module uses Maven as project management. On a Maven-enabled system, building the DSB is as easy as launching the Maven command with the release profile flag to generate the final binary:

```
mvn install -Pdistributions
```

As a result, the DSB binary is available under the *distribution/dsb-distribution/target* folder.

3.3.2 Install

Independently of the method used to get the DSB binary (compile sources as described in the previous section or by direct download), installing the DSB means un-archiving the DSB ZIP file.

In the next sections, we assume the DSB is installed under the *$DSB* folder.

3.3.3 Configure

The DSB is the main integration module of the project and configuration is an important step. All the configuration files are located under the *$DSB/conf* folder. The current section only provides details for the main configuration files which may need some to be updated according to PLAY needs. Other files can be left as provided.

3.3.3.1 Topology

The DSB is distributed by nature. It means some can launch several instances and connect them together to extend integration capabilities and locate nodes closer to business services.

In the current status, we focus on the simplest integration case and run one DSB node. The *server.properties* file is a standard java properties file and can be left as it is provided. When installing the DSB node on a public host, you may require to update the *topology.xml* file to set the IP address so that the node is reachable and all internal modules use the same address. This is defined in the *host* XML element. You may replace 'localhost' with the right value.

3.3.3.2 Business services

Even if the DSB provides a management API, it can be useful to bind some services at configuration time. This is possible by using the *services2bind.cfg* properties file. To provide a list of SOAP-based services to bind, you have to define the soap.list CSV value. For example, to bind two Web services, provide their WSDL URLs like:

```
soap.list=http://host:port/Foo?wsdl,http://host:port/Bar?wsdl
```

3.3.3.3 Event Consumer

The DSB can be preconfigured to subscribe to WSN event producers at startup. This is defined in the *consumer.cfg* file where parameters are:

- **consumerReference**: The endpoint which will be notified when a new notification is available in the topic ie the subscriber.
- **producerReference**: The endpoint to send subscription to.
- **topicName, topicURI, topicPrefix**: The WSN topic to subscribe to.

In order to be able to create lists, the previous parameters have to be prefixed with an identifier. For example, to create two subscribers, simply create entries:

```
subs0.consumerReference=
subs0.producerReference=
subs0.topicName=
…
subs1.consumerReference=
subs1.producerReference=
subs1.topicName=
…
```

3.3.3.4 Event Subscribers

The DSB can be preconfigured to add subscribers on its local topics at startup. This is defined in the *subscribers.cfg* configuration file where parameters are:

- **consumerReference**: The endpoint of the subscriber. A notification will be sent to this endpoint when a message is published in the topic defined below.
- **topicName, topicURI, topicPrefix**: The local WSN topic to add consumer to.

As defined in the previous section, you can define list by prefixing parameters.

3.3.3.5 Topics definition

The supported topics are defined by configuration in the XML files under the *topics* folder. Topics are separated into business and kernel ones. We focus on the business ones which are defined in the *topics/business-topicset.xml* file. All the supported topics must be defined in this file following the given template: The root element is the TopicSet, all children are the topics which are supported by the DSB. If some subscribes to a topic which is not defined here, the subscription will be rejected and an exception will be raised.

For example, we define the *NiceTempStream*, *NiceWeatherStream*, *FacebookStatusFeed* and *PachubeFeed* topics like that:

```
<wstop:TopicSet xmlns:wstop="http://docs.oasis-open.org/wsn/t-1"
        xmlns:s="http://streams.event-processing.org/ids/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:dcterms="http://purl.org/dc/terms/" xmlns:xhtml="http://www.w3.org/1999/xhtml#">
        <s:NiceTempStream wstop:topic="true"
                xhtml:icon="http://www.ville-nice.fr/design/standard/images/favicon.ico"
                dcterms:title="Nice Temperatures" dcterms:description="A stream of temperature readings."></s:NiceTempStream>
        <s:NiceWeatherStream wstop:topic="true"
                xhtml:icon="http://www.ville-nice.fr/design/standard/images/favicon.ico"
                dcterms:title="Nice aggregated Weather" dcterms:description="A stream of complex events."></s:NiceWeatherStre
        <s:FacebookStatusFeed wstop:topic="true"
                xhtml:icon="https://s-static.ak.facebook.com/rsrc.php/yi/r/q9U99v3_saj.ico"
                dcterms:title="Facebook Status Feed" dcterms:description="A stream of Facebook Wall updates."></s:FacebookSta
        <s:PachubeFeed wstop:topic="true"
                xhtml:icon="http://pachube.com/favicon.ico" dcterms:title="Pachube Stream"
                dcterms:description="A stream of Pachube events."></s:PachubeFeed>
</wstop:TopicSet>
```

Figure 1 - DSB Topics definition

3.3.4 Run

Once the DSB is configured by following instructions from the previous section, running the DSB is possible by launching the *$DSB/bin/start.sh* shell script from the DSB installation folder. There are several additional ways to start the DSB with some options:

1. *'start.sh'*: Starts the DSB in the foreground without any interaction. Use CTRL+C to shutdown it.
2. *'start.sh –C'*: Starts the DSB in console mode (preferred way in development mode). Once the DSB is fully started, the user can interact with the DSB and retrieve some useful information (type h for help).
3. *'start.sh –D'*: Starts the DSB in daemon mode (in the background).

3.4 Governance

The governance is a Java-based runtime. It runs on any Java Virtual machine version 6 or later. The binary distribution of the governance runtime can be downloaded from http://research.petalslink.org/display/easiergov/Binaries or generated from sources as described in the next subsection.

The governance runtime needs 25 Mb of hard disk space and at least 1 Gb of memory to run.

3.4.1 Generating binary from sources

The source code is hosted on a SVN repository at https://svn.petalslink.org/svnroot/trunk/research/dev/experimental/easiergov/ and can be retrieved using a SVN client:

```
svn co
https://svn.petalslink.org/svnroot/trunk/research/dev/experiment
al/easiergov/
```

If credentials are requested by the system, use 'anonymous' as login and password (without the quotes) to have read access.

The governance module uses Maven as build system, you can build it with the following command:

```
mvn install
```

Once the build is complete, the governance distribution is available under the *gov-ws-distribution/target* folder.

3.4.2 Install

The governance module comes with a GUI installer to ease the installation procedure. Once the JAR file is generated or downloaded, launch it from the command line (*java -jar*) or double-click it and follow the instructions.

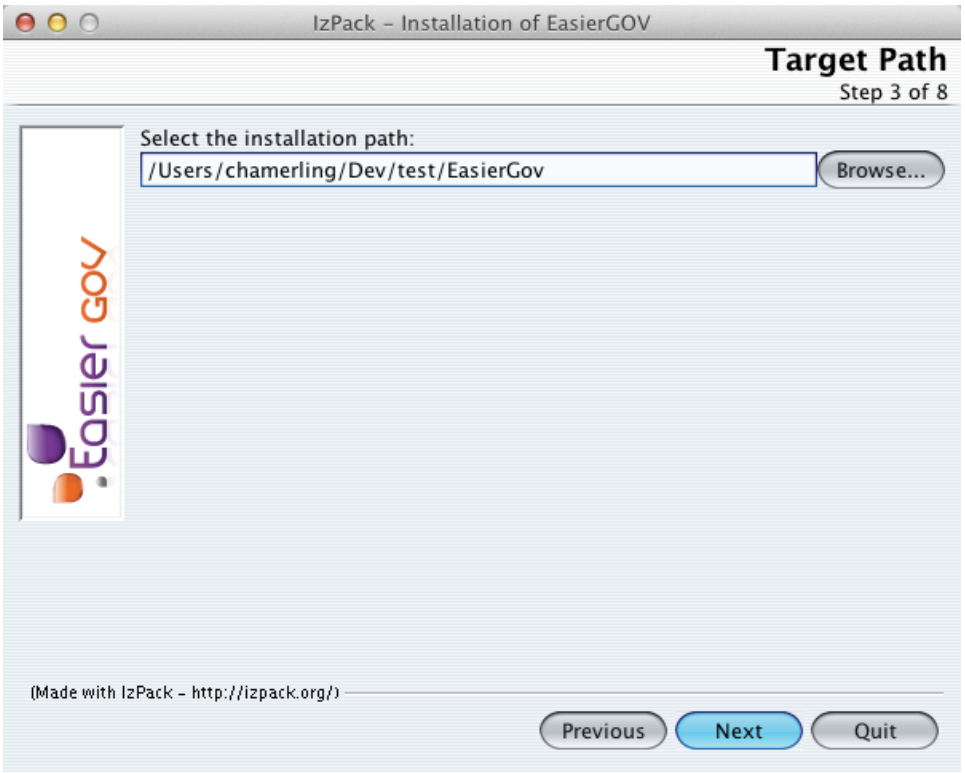

Figure 2 - Governance installation

In the next sections, we assume that the monitoring runtime is installed in a folder named $GOVERNANCE.

3.4.3 Configure

The monitoring configuration file is located under the *$GOVERNANCE/bin/* folder and named *config.properties*. Available parameters are:

- **host**: The local hostname
- **port**: The port the monitoring runtime is exposing its Web services
- **name**: The name of the runtime. Can be left as provided
- **namespace**: The namespace used by the runtime. Can be left as provided.

3.4.4 Run

The governance runtime can be started using the *startup.sh* shell script located under the *$GOVERNANCE/bin/* folder.

When launched, user can interact with the runtime and get back useful information with the info or help commands: ‘i’ or ‘h’.

3.5 Monitoring

The monitoring module is a Java-based runtime. It runs on any Java Virtual machine version 6 or later. The binary distribution of the monitoring runtime can be downloaded from http://research.petalslink.org/display/easierbsm/Binaries or generated from sources as described in the next subsection.

The monitoring runtime needs 30 Mb of hard disk space and at least 1 Gb of memory to run.

3.5.1 Generating binary from sources

The source code is hosted on a SVN repository at https://svn.petalslink.org/svnroot/trunk/research/dev/experimental/easierbsm/ and can be retrieved using a SVN client:

```
svn co
https://svn.petalslink.org/svnroot/trunk/research/dev/experiment
al/easierbsm/
```

If credentials are requested by the system, use 'anonymous' as login and password (without the quotes) to have read access.

The monitoring module uses Maven as build system; you can build it with the following command:

```
mvn install
```

Once the build is complete, the monitoring distribution is available under the *bsm-distribution/target* folder.

3.5.2 Install

The monitoring module comes with a GUI installer to ease the installation procedure. Once the JAR file is generated or downloaded, launch it from the command line (*java -jar*) or double-click it and follow the instructions.

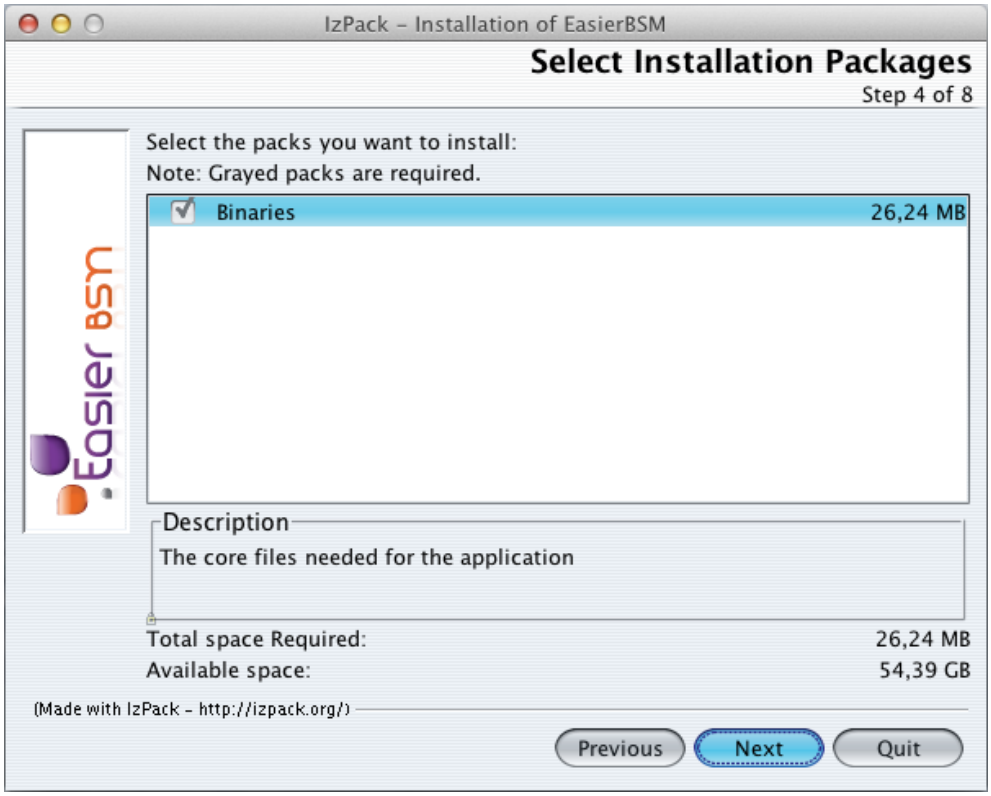

Figure 3 - Monitoring installation

In the next sections, we assume that the monitoring runtime is installed in a folder named $MONITORING.

3.5.3 Configure

The monitoring configuration file is located under the *$MONITORING/bin/* folder and named *config.properties*. Available parameters are:

- **host**: The local hostname
- **port**: The port the monitoring runtime is exposing its Web services

3.5.4 Run

The monitoring runtime can be started using the *startup.sh* shell script located under the *$MONITORING/bin/* folder.

When launched, user can interact with the runtime and get back useful information with the info or help commands: ‘i’ or ‘h’.

3.6 SAN Editor

SAN Editor is the dedicated graphical tool for designing SANs, performing context model specialisations based on the application scenario and stating the necessary queries to events for extracting contextual information. Using this editor, SANs are visualised in two different tree-like graphical representations and can be exported in an executable format, i.e. SAN language (described in D4.1.2). SAN trees created using SAN editor are currently used by Events Subscriptions Recommender (ESR) software component.

3.6.1 Install

This section is giving a short description of where to find and how to install the SAN Editor. The SAN Editor is a web-based application which runs within any flash-enabled web browser. The latest version SAN Editor can be found online at: http://imu.ntua.gr/software/san-editor/latest/ (Username: Play, Password: Play).

The flash player can be downloaded from Adobe at http://www.adobe.com/software/flash. The table below contains all the available flash player versions.

Platform	Browser	Player version
Windows	Internet Explorer (and other browsers that support Internet Explorer ActiveX controls and plug-ins)	11.1.102.63
Windows	Firefox, Mozilla, Netscape, Opera (and other plugin-based browsers)	11.1.102.63
Macintosh - OS X	Firefox, Opera, Safari	11.1.102.64
Linux	Mozilla, Firefox, SeaMonkey	11.1.102.63
Windows	Chrome	11.1.102.63
Macintosh - OS X	Chrome	11.1.102.64

Linux	Chrome	11.1.102.63
Solaris	Mozilla	11.1.102.63

Table 2 - Flash and Browser compatibility

3.6.2 Configure

SAN Editor doesn't require any specific configuration before running.

3.6.3 Run

SAN editor can run by opening the **MAIN.html** using any flash-enabled web browser. Then the user can create and manage any number of different SAN trees in a user-friendly and graphical way.

3.7 Event Subscriptions Recommender

3.7.1 Background

Events Subscription Recommender (ESR) version 1.0, has been developed during PLAY project as one of its value added services. Its purpose is to receive and process events originating from PLAY's Event Marketplace and based on predefined scenarios, to generate and deliver recommendations for subscriptions to event streams, to consumers (either human or software) interested, in a timely fashion. Alternatively ESR might send its recommendations back to Event Marketplace. ESR uses SAN trees created using the SAN editor discussed above.

Requirements

ESR requires approximately 50MB of disk space plus 1MB extra for source code and examples. However depending on the scenario, additional disk space will be required, either as temporary data, or for input or output or log files, or even in OS swap files.

In terms of memory at least 500MB of RAM are required. It is recommended that 2GB of RAM to be present in the System.

ESR can operate both in a single CPU or multiple CPU systems. Scenarios generating several entities and root goals will be benefiting from multiple CPUs.

No Graphical User Interface (GUI) is required. It can be operated from a command line interface (e.g. console, telnet sessions).

ESR requires Java version 1.6.0 or later in order to work properly.

Briefly. SAN engine requirements:

Requirement	Value
Disk space (installation)	50MB + 1MB for source code and examples
Disk space (usage)	100MB or more, depending on the scenario
RAM (min / recommended)	500MB / 2GB
CPU (actual or cores)	1 or more
GUI	Not required
Java	Standard. Version 1.6.0 or later

Table 3 - SAN engine requirements

3.7.2 Install

3.7.2.1 Download

ESR is distributed as an archive, either in ZIP format or RAR format. It can be downloaded from IMU web site:

http://imu.ntua.gr/software/ESR/latest/esr-v1.0.rar

3.7.2.2 Uncompress archive

The archive containing ESR can be uncompressed using an appropriate application. For example, unzip or gunzip utility on UNIX/LINUX-based systems, or PKZIP on MS Windows-based systems. This archive can be uncompressed at any location in a local or network disk and directory, where user has read/write access to.

3.7.2.3 Directory structure

Uncompressing the archive will create the following file and directory structure.

File or Directory	Description
san-egine	The home directory of the installation.
├ bin │	contains Unix/Linux scripts and BAT files that launch ESR for various configuration file.
├ classes	Contains the compiled code of ESR.
├ config	Contains configuration files of the various examples.
│ └ configuration.txt	It is the default configuration file. Use it as template.
├ examples	Various examples in RDF/N3 file format.
├ lib	Contains java libraries from third parties.
├ src	Contains the source code used
│ ├ org.iccs	The source code of ESR.
│ ├ org.oasis_open	Imported source code for OASIS WS-Notif messages
│ └ org.ow2	Imported source code for OASIS WS-Notif messages
└ tmp	Temporary files can be stored here

Table 4 - SAN directory structure

3.7.2.4 Modify scripts

Default ESR installation provides configuration and scripts using default values. They assume that ESR installation / home directory is at **"T:\"**. If needed, end users can modify scripts to suit their needs. This is required if ESR is uncompressed in a location other than **"T:\"**. The scripts are UNIX Bash files and MS Windows BAT files, and they are located in the **bin** directory. Examples: **run.sh** or **run.bat** scripts.

It is worth-mentioning that the use of scripts in **bin** directory is optional. Users might also start ESR using java commands (i.e. **java** or **javaw**). Please see run.sh or run.bat in **bin** directory for a template.

3.7.2.5 Usage

Users can enter the ESR Command-Line Interface (CLI) by giving the following command at the console prompt:

```
java -classpath ".;classes;lib\*;lib\lib-client\*;lib\lib-
webservices\*"  org.iccs.san.engine.launch.CLI
```

or simply giving:

```
run
```

This command will start the interactive CLI shell of ESR. In the prompt the user can write **help** to see the help screen.

```
write 'quit' to exit or '?' for more commands  :  help
```

The available commands are listed in the following table. Their shortcuts are also given:

Command	Short	Description
context	**x**	Prints current context(s) values. Usage: **context** prints all contexts contents **context list** lists all contexts URIs (unique identifiers) **context list local** lists all local contexts URIs **context list entity** lists all entity URIs

		context <URI> prints contents of the specified context
cepats	**p**	Lists the deployed CEPATs with status information
continue	**c**	Resumes SAN execution after a BREAK decorator
help	**?**	Help screen
quit	**q**	Exits ESR
reload	**r**	Reloads configuration file by restarting engine. Contexts are preserved
source	**s**	Changes the source file and restarts engine. Contexts are preserved
event	**e**	A) Publishes an event, using the underlying CEP engine facility. Usage: **event <ID> <NAMESPACE> <TOPIC> <PREFIX> <EVENT_CONTENT>** ▪ if FileCEPEngine is used: It writes an event file in 'locks' directory ▪ if DsbCEPEngine is used: It publishes an event to DSB B) Queries/sets the online status of CEP engine. Usage: **event [on|off|status]**
entities	**n**	Lists the active entities
next	**.**	Replay next event. This command is valid only if DsbCEPEngine is configured and only if replaying is in manual mode.

Table 5 - SAN commands

At this mode user can instruct ESR to load a use case file and start executing it.

```
source <path_to_use_case_file>
```

This command will restart ESR and start executing the file instructions. An alternative way to instruct ESR execute a use case file is through command line:

```
run –source <path_to_use_case_file>
```

3.7.3 Configure

ESR configuration file is stored in a plain text file using a name-value pair format. In particular, it uses the format of a property file of Java. Configuration file must specify all necessary settings required from ESR in order to configure its subsystems as well as any use case specific data.

When starting ESR checks the **–config** command line switch in order to take the name of the configuration file to be used. If not present it checks for file **"configuration.txt"** in the current directory or in **"config"** directory.

3.7.3.1 Configuration file format

At each line of configuration file, one name-value pair can be specified. The name and value parts are separated with a white space character (space or tab), or an equal sign **"="** or a colon **":"**. Subsequent occurrences are considered as part of the value. Lines starting with hash mark **"#"** are comments and they are ignored.

In order to avoid confusion when two or more parameters need to use the same name, a suitable naming convention can be used. The recommended style, which is also used from ESR itself for configuring its subsystems, is a dot-delimited, directory-like approach. In this approach every subsystem and component in the containment hierarchy of a parameter should be named. For instance **'context.db.connect-string'**.

Some names are reserved for ESR configuration purposes. Apart from them any other name can be used for use case purposes.

3.7.3.2 Reserved names

Each ESR subsystem uses certain names in configuration file in order to get its settings. A list of them is given below, along with the allowed values and defaults, and a short description.

Name	Values / Default	Description
engine.class	Java class / org.iccs.san.engine.naive.NaiveSANEngine	SAN engine implementation class
engine.auto-start-root-goals	yes, no / yes	Whether starting root goals on ESR launch
engine.naive.exit-delay	Positive integer / 1000	Delay before exiting ESR in milliseconds
repository.class	Java class / org.iccs.san.repository.sesame.SesameSANRepository	SAN repository implementation class
repository.source	Use case file / %source%	File containing the use case entities and SANs By default it is? required as a command-line value
context.class	Java class / org.iccs.san.context.InMemoryContext	Context subsystem implementation class. Default: InMemmoryCtx Alternative: DatabaseContext
context.dispose-contexts	yes, no / no	Dispose context objects when the owning SAN or entity ends?
context.db.connect-string	<string>	Connect string to MySQL
context.db.username	<string>	Username
context.db.password	<string>	Password
context.db.query-file	<string>	File containing SQL queries (Provided with DatabaseContext impl)
cep-engine.class	Java class / org.iccs.san.cep.FileCEPEngine	Events engine implementation class. Default: FileCEPEngine Alternative: DsbCEPEngine
cep-engine.file.cmd	System command / T:/bin/cep-file/gen-events.bat %s.txt	Script executed when an event should be sent.
cep-engine.file.events-dir	Directory / tmp	Directory of File CEP eng. command files
cep-engine.dsb.PUBLISH_ENDPOINT	URL	DSB publish endpoint
cep-engine.dsb.SUBSCRIBE_ENDPOINT	URL	DSB subscribe endpoint
cep-engine.dsb.TOPICS_ENDPOINT	URL	DSB topics endpoint
cep-engine.dsb.NOTIFY_ENDPOINT	URL	Local notifications endpoint
cep-engine.dsb.NOTIFICATION_PRODUCER_ENDPOINT	URL	Local notifications producer endpoint
cep-engine.dsb.IP_ADDRESS_FILTER	Part of an IP address	IP address filter used if system should resolve IP address automatically
cep-engine.dsb.IP_ADDRESS	IP address or NO-NETWORK	IP address to bind local endpoints. NO-NETWORK can be used when in replay mode.
cep-engine.dsb.SUBSCRIPTION_METHOD	CFX or CLIENT_LIB	Subscription method
cep-engine.dsb.replay-file	File or Directory	Events file (in XML) or directory

Name	Values / Default	Description
cep-engine.dsb. replay-delay	Positive integer	Delay in milliseconds between event sendings
cep-engine.dsb. replay-start-delay	Positive integer	Delay in milliseconds before start reading events
cep-engine.dsb. replay-mode	BATCH or MANUAL	In Batch mode no user intervention is needed
cep-engine.dsb. replay-exit-after	yes, no / no	Exit ESR after replay completion
cep-engine.dsb. replay-allow-send-events	yes, no / no	Send events to DSB while in replay mode
cep-engine.dsb. DEBUG_EVENTS	yes, no / no	Print all events received
cep-engine.dsb. PRINT_XML	yes, no / no	Print payload of events delivered to a SAN.
io-system.class	Java class / no default	I/O system class. If none specified, output goes to console
io-system.file	File	File where output and errors are recorded.
helper.action.use-gui	yes, no / yes	Display Javascript alert and confirm dialogs
stats.file	File ext.	Statistics file
stats.subscriptions.file	File ext.	Subscriptions file

Table 6 - ESR configuration parameters

3.7.3.3 Additional configuration files

For better housekeeping it is possible to split a configuration file into several files. This is especially useful when a group of settings is common to many applications, for instance the DSB endpoints. The master (application) configuration file must specify the other configuration files using the **"@include"** directive. Their contents are imported and inserted at the location of the directive.

```
@include  config/extra.settings.txt
```

The "@include" directive can also be the value part in a name-value pair. In this case all names of the imported file are prefixed with the name just before directive. For example:

```
cep-engine.dsb :      @include  config/dsb.properties
```

Assuming the contents of **config/dsb.properties** are:

```
PUBLISH_ENDPOINT=http://.../petals/pubSrv
SUBSCRIBE_ENDPOINT=http://.../petals/subSrv
TOPICS_ENDPOINT=http://.../petals/topicsSrv
```

Then what is really imported in master configuration file is:

```
cep-engine.dsb.PUBLISH_ENDPOINT = http://.../petals/pubSrv
cep-engine.dsb.SUBSCRIBE_ENDPOINT = http://.../petals/subSrv
cep-engine.dsb.TOPICS_ENDPOINT = http://.../petals/topicsSrv
```

3.7.4 Run

In order to run ESR a command-line interface (e.g. a console, telnet sessions) should be opened, and a change to san-engine directory must take place:

type: `bin\setenv` in Windows
or `source bin/setenv` in Unix/Linux

and type:

```
run –config <path_to_config_file> –source <path_to_N3_file>
```

3.8 AIS Adapter

To run AIS adapter binaries, Java 6 or greater and MongoDB is required.

3.8.1 Installation

- Download AIS adapter from: http://imu.ntua.gr/software/play-ais-adapter/latest/AISAdapter.zip
- Extract the archive AISAdapter.zip
- Download MongoDB from http://www.mongodb.org/
  - Extract MongoDB
  - Create the directory $MDBDATA/aisdb, where $MDBDATA the desired data directory for MongoDB

3.8.2 Configure

In order to change the configuration of the AISAdapter, edit the file *AISAdapter/bin/app.properties*:

- Define the PLAY DSB properties
  - `DSB_IP=46.105.181.221:8084`
  - `PUBLISH_ENDPOINT= http://#DSBIP#/petals/services/NotificationConsumerPortService`
- Define the host and port of AIShub.net (or an AISHub tcp relay)
  - `AIS_TCP_HOST_PORT=147.102.23.45:4008`
- Filter sent proximity events by maximum distance from each vessel
  - `#Maximum distance for proximity events (in meters) (1 nautical mile =1852m)`
    `MAX_DISTANCE=9000`
- Filter out events outside a box defined by maximum and minimum latitude/longitude pairs
  - `#Filter vessels within a box (Greece lat 32-42, lon 20-30)`
    `MINLAT=32.0`
    `MAXLAT=42.0`
    `MINLON=20.0`
    `MAXLON=30.0`
- Define a delay between publishing events to DSB
  - `#Time to sleep after each event publish (in ms)`
    `SLEEPTIME=100`

3.8.3 Run

- First step is to start MongoDB. Go to the MongoDB bin directory and run:

```
mongod -dbpath $MDBDATA/aisdb
```

- Start the AISAdapter from the AISAdapter directory :

```
java -cp "bin;lib\*;lib-play\*;lib-sesame\*;extlib\*"
org.iccs.play.ais.AisRDFAdapter
```

4 Event Source Adapters

In PLAY we provide several event sources including their necessary adapters. The sources are chosen to augment the Event Marketplace with events from the Social Web as well as the Internet of Things.

Namely these sources include: (1) A **Facebook App** which a user can allow to notify all Facebook Wall updates as PLAY events. (2) A **Pachube**[3] adapter which can subscribe to sensor readings and similar events from the Internet of Things and can flexibly be transformed into PLAY events. (3) A **Twitter** adapter which uses the Twitter API[4] to receive Tweets and convert them to PLAY events. These adapters were first described in deliverable D2.5.1, Section 4.2.3., and we will go into some detail in the following.

Moreover, at the end of this section we describe an output adapter. Its purpose is to consume events and format them as plain RDF streams which can be subscribed to in a very simple way by dereferencing (e.g. in your browser) the stream ID which is a URL.

4.1 Facebook Adapter

The environment for our system is based on Java EE standards with some open source APIs. Apache Maven[5] is used for the project management. The applications are deployed on Tomcat 6.

We divide our implementation into three modules. First, subscribing and retrieving the information from Facebook. Second, transforming this information to RDF events by using RDFReactor API. Third, using WS-Notification publish/subscribe for event streaming.

First, a Tomcat servlet is created for retrieving information and creating events. This application registers to Facebook so that it can receive events in real-time. Whenever authenticated Facebook users post something on their Facebook Wall, a Facebook real-time notification is sent to our servlet. The servlet then fetches the necessary data which is not part of the Facebook notification such as the user's location, the message content, etc.

Then, the data is transformed into RDF events. Those events are sent to the Distributed Service Bus (DSB) afterwards for use in the PLAY platform.

An example event from the Facebook adapter is depicted in Listing 1 - Facebook Event example

. It demonstrates the use of all attributes currently in the schema. Some attributes are in the default namespace (e.g., `:status`), some are in the namespace "`user:`"[6] defined by the Facebook Graph API (e.g., `user:id`).

[3] Pachube: Web portal to connect sensor data: www.pachube.com

[4] Twitter API: https://dev.twitter.com/

[5] Maven build tool: http://maven.apache.org/

[6] Facebook and RDF: http://vanirsystems.com/blog/2011/10/13/a-quick-post-on-facebook-and-linked-data/

```
@prefix :        <http://events.event-processing.org/types/> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix user:    <http://graph.facebook.com/schema/user#> .

<http://events.event-processing.org/ids/e6575098690361423887#event>
    a       :FacebookStatusFeedEvent ;
    :endTime "2012-05-25T11:23:55.735Z"^^xsd:dateTime ;
    :status "I bought some JEANS this morning" ;
    :stream <http://streams.event-processing.org/ids/FacebookStatusFeed#stream> ;
    user:id "100000058455726" ;
    user:link <http://graph.facebook.com/roland.stuehmer#> ;
    user:location "Karlsruhe, Germany" ;
    user:name "Roland Stühmer" .
```

Listing 1 - Facebook Event example

4.2 Pachube Adapter

In PLAY we develop a Pachube[7] adapter. Its purpose is to subscribe to sensor readings and similar events from the Internet of Things. Using the adapter, such events can flexibly be transformed into PLAY events.

To connect Pachube to the PLAY Event Marketplace we have implemented a Java servlet running on Tomcat 6. The servlet is exposed to the Web in order for Pachube to invoke it whenever there is new data using WebHooks[8]. A WebHook is an HTTP callback: an HTTP POST that occurs when something happens. When the servlet is invoked, it parses the data from Pachube, converts it to RDF and attaches it to a PLAY event using an RDFReactor class specific to Pachube events. The data from Pachube arrives as non-semantic JSON data. We are using a so-called lifting to create meaningful RDF from the structured JSON data. The lifting is implemented as a SPARQL CONTRUCT query. First, JSON is converted to "naïve" RDF by replicating only the structure, not the semantics. Then, CONSTRUCT queries are used like an RDF to RDF transformation, where meaningful RDF properties are introduced from well-known schemas. These properties replace the merely structural ones. This is done in order to make the RDF more usable as semantic events. Listing 2 shows some *structure* RDF before translation.

[7] Pachube: Web portal to connect sensor data: www.pachube.com

[8] WebHooks, HTTP callbacks: http://www.webhooks.org/

```
<http://events.event-processing.org/ids/pachube74771736682889971#event>
    a       :PachubeEvent ;
    :endTime "2012-05-25T12:39:27.503Z"^^xsd:dateTime ;
    :stream <http://streams.event-processing.org/ids/PachubeFeed#stream> ;
    :rawData
          [ <http://www.linkedopenservices.org/ns/temp-json#creator>
                "https://pachube.com/users/rossoreed" ;
            <http://www.linkedopenservices.org/ns/temp-json#datastreams>
                [ <http://www.linkedopenservices.org/ns/temp-json#at>
                      "2011-10-12T14:13:16.253064Z" ;
                  <http://www.linkedopenservices.org/ns/temp-json#current_value>
                      "7.159" ;
                  <http://www.linkedopenservices.org/ns/temp-json#id>
                      "sd4" ;
                  <http://www.linkedopenservices.org/ns/temp-json#max_value>
                      "21.626" ;
                  <http://www.linkedopenservices.org/ns/temp-json#min_value>
                      "0.0" ;
                  <http://www.linkedopenservices.org/ns/temp-json#tags>
                      [ <http://www.linkedopenservices.org/ns/temp-json#at>
                            "Acc Home Power (24hr reset)"
                      ] ;
                  <http://www.linkedopenservices.org/ns/temp-json#unit>
                      [ <http://www.linkedopenservices.org/ns/temp-json#label>
                            "kWh" ;
                        <http://www.linkedopenservices.org/ns/temp-json#symbol>
                            "kWh"
                      ]
                ] .
```

Listing 2 - Pachube Event Example about Power Consumption

4.3 Twitter Adapter

The Twitter adapter uses the Twitter API[9] to receive Tweets and convert them to PLAY events. To connect to the Twitter API we have implemented another dedicated Tomcat servlet. It makes heavy use of the Twitter4J[10] library, an unofficial Java library for the Twitter API. The Twitter API "allows high-throughput near real-time access to various subsets of public and protected Twitter data"[11]. Public statuses are available from all users, filtered in various ways: By user id, by keyword, by random sampling, by geographic location, etc.

Listing 3 shows an example Twitter event displaying properties from our schema. As a best practice in ontology design we not only define our own schema but are reusing existing schemas to increase interoperability with other software and increase semantic understanding of our data. Thus, our schema uses event properties from the namespace `sioc:` in the SIOC Ontology[12] to describe user generated content on the Web 2.0. Moreover, we reuse properties from the W3C Basic Geo Vocabulary[13] in the namespace `geo:`.

[9] Twitter API: https://dev.twitter.com/

[10] Twitter4J, an unofficial Java library for the Twitter API: http://twitter4j.org/

[11] Twitter API, recently renamed to Streaming API: https://dev.twitter.com/docs/streaming-api

[12] SIOC Core Ontology Specificationhttp://sioc-project.org/ontology

[13] Basic Geo (WGS84 lat/long) Vocabulary: http://www.w3.org/2003/01/geo/

```
@prefix :        <http://events.event-processing.org/types/> .
@prefix geo:     <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix sioc:    <http://rdfs.org/sioc/ns#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://events.event-processing.org/ids/twitter3904330550437717 5#event>
    a       :TwitterEvent ;
    :endTime "2011-06-02T15:06:45.000Z"^^xsd:dateTime ;
    :followersCount  "10"^^xsd:int ;
    :friendsCount "1"^^xsd:int ;
    :isRetweet "false"^^xsd:boolean ;
    :screenName "softamo" ;
    :stream <http://streams.event-processing.org/ids/TwitterFeed#stream> ;
    :twitterName "Sergio del Amo" ;
    :location [
            geo:lat    "43.616231774652796"^^xsd:double;
            geo:long  "7.053824782139356"^^xsd:double
        ] ;
    sioc:content "Anímate a participar en el programa @wayra de Telefónica y
        consigue apoyo integral para tu proyecto http://bit.ly/k84qgN #iniciador" .
```

Listing 3 - Twitter Event Example

4.4 Linked Data Streaming Adapter

The Linked Data Streaming Adapter is an output adapter, meaning it can be used by clients to obtain events from the PLAY platform. Events can be selected by stream. If a client knows a stream ID (e.g. from a historic event or from the WebApp catalogue of streams) the client can get a real-time feed of all events in this stream.

The purpose of the Linked Data Streaming Adapter is to advance the principles of Linked Data towards *real-time* Linked Data. Linked Data is the methodology of publishing structured (static) data as RDF and to interlink the data to make it more useful. Examples of Linked Data are movies and their globally unique identifiers which can be found on-line. These identifiers are useful in identity management on the Web. Publishing Linked Data in general is done using four principles[14]:

1. Use URIs to identify things.
2. Use HTTP URIs so that these things can be referred to and looked up ("dereferenced") by people and user agents.
3. Provide useful information about the thing when its URI is dereferenced, using standard formats such as RDF/XML.
4. Include links to other, related URIs in the exposed data to improve discovery of other related information on the Web.

This was described and implemented for static data in RDF but not for streaming data. Such data could also profit from the aforementioned principles and the principles apply just as well. However, there are no standards and no implementations of streaming data. For this adapter, we are developing an RDF Streaming API to adapt the four Linked Data principles to real-time applications. At the same time the PLAY event

[14] Berners-Lee, Tim (2006). Linked Data. http://www.w3.org/DesignIssues/LinkedData.html

format is built on top of RDF so that our data modeling language fits seamlessly with the data dissemination.

We are currently implementing the adapter using the *Play framework*[15] 2.0.

[15] Play framework: http://www.playframework.org/ (not affiliated with the PLAY FP7 project)

5 Applications

In this chapter we highlight a selection of scenarios implemented on top of the PLAY platform. The scenarios illustrate the platform features, the event sources and the event patterns used in realizing such applications.

5.1 Facebook "Jeans Example"

End-users get the option to install a Facebook application [1] developed within the PLAY project. The goal of this application is to create PLAY-events from the Facebook platform. The current choice is to create these events for each Facebook Wall update i.e. each time a Facebook user updates his status. The users can then go to the PLAY Web application and see the events arriving in real-time as well as results from a running event pattern which is processing these Facebook events.

The pattern we want to introduce currently detects situations where three people on Facebook mentioned the same keyword "Jeans" within the last 30 minutes. See Listing 4. The pattern syntax is described in deliverable D3.2, Section 2.4.

Events are then sent from the Facebook application to the PLAY platform using the PLAY API and goes through the bus, the Event Cloud and the DCEP. The PLAY platform processes the events and then reacts based on complex patterns defined and running in the platform. As a result, reactions to a complex pattern are the display in the Web browser (real-time push with AJAX technology), as an e-mail alert (by supplying an address) or by getting notifications in a long-standing way without having to look at the Web application continuously.

```
# $Revision: 32522 $
# $Id: play-epsparql-m12-jeans-example-query.eprq 32522 2012-05-23 12:32:36Z stuehmer $

PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX user:   <http://graph.facebook.com/schema/user#>
PREFIX s:      <http://streams.event-processing.org/ids/>
PREFIX :       <http://events.event-processing.org/types/>

CONSTRUCT {
   :e rdf:type :FacebookCepResult .
   :e :stream <http://streams.event-processing.org/ids/FacebookCepResults#stream> .
   :e user:name ?friend1 .
   :e user:name ?friend2 .
   :e user:name ?friend3 .
   :e :discussionTopic ?about1 .
   :e :discussionTopic ?about2 .
   :e :discussionTopic ?about3 .
}
WHERE {
   WINDOW {
      EVENT ?id1 {
         ?e1 rdf:type :FacebookStatusFeedEvent .
         ?e1 :stream <http://streams.event-processing.org/ids/FacebookStatusFeed#stream> .
         ?e1 :status ?about1 .
         ?e1 :name ?friend1 .
         }
         FILTER fn:contains(?about1, "JEANS")
      SEQ
      EVENT ?id2 {
         ?e2 rdf:type :FacebookStatusFeedEvent .
         ?e2 :stream <http://streams.event-processing.org/ids/FacebookStatusFeed#stream> .
         ?e2 :status ?about2 .
         ?e2 :name ?friend2 .
         }
         FILTER fn:contains(?about2, "JEANS")
```

```
        SEQ
        EVENT ?id3 {
           ?e3 rdf:type :FacebookStatusFeedEvent .
           ?e3 :stream <http://streams.event-processing.org/ids/FacebookStatusFeed#stream> .
           ?e3 :status ?about3 .
           ?e3 :name ?friend3 .
           }
           FILTER fn:contains(?about3, "JEANS")
    } ("P30M"^^xsd:duration, sliding)
}
```

Listing 4 - EP-SPARQL Query for "Jeans" Example

At any time, users can compose new queries using the Web application, and can be notified in real-time of complex events filling that pattern.

5.2 Contextualized Latitude

For this scenario we combine information from the social media channels (currently Facebook and Twitter) and the smartphone events (currently Android based, GPS position, incoming calls, missing calls and outgoing calls). Note that the list of events and situations to be demonstrated will be larger since the development of the PLAY Platform and the connection to various event sources is ongoing.

Fehler! Verweisquelle konnte nicht gefunden werden. illustrates the general flow of events in the scenario. Following are illustrations of the situations which this scenario demonstrates.

Figure 4 – General Scenario and Event Flow of Contextualized Latitude

1) A general example is defined for the scenario, that is based on some time interval and simple notification – e.g. if the user was calling 2 times within 10 min without success, then remember (send notification) her/him to call again after 60min from the last try

2) The customer (caller) tries to call a friend. The friend (callee) does not answer for whatever reason. The PLAY platform is aware of this (after a sequence of calls) and detects that this friend is online on Facebook (or have sent recently tweets on twitter). PLAY informs him about this and automatically opens Facebook or twitter

3) The customer is in the taxi and calls a friend. They will meet together when the taxi is at destination, but the friend is located in a crowd of people (in a stadium) and so does not answer. The friend has not started Facebook nor Twitter, but Customer and Friend are sending location updates to PLAY. After 2 missed calls, PLAY platform reacts and proposes to use the recent location updates to localize my friend, and trace the route between customer and friend.

The first iteration of one of the event patterns is shown in Listing 5. The pattern combines events from streams about Twitter and Geolocation Updates into a new stream “ContextualizedLatitudeFeed”.

```
# $Revision: 32522 $
# $Id: play-epsparql-contextualized-latitude-01-query.eprq 32522 2012-05-23 12:32:36Z stuehmer $

PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX uctelco: <http://events.event-processing.org/uc/telco/>
PREFIX :       <http://events.event-processing.org/types/>

CONSTRUCT {
   :e rdf:type :ContextualizedLatitudeEvent .
   :e :stream <http://streams.event-processing.org/ids/ContextualizedLatitudeFeed#stream>
         :e :location [ geo:lat ?Latitude1; geo:long ?Longitude1 ] .
         :e uctelco:phoneNumber ?bob .
         :e uctelco:phoneNumber ?alice .
   :e :message "Alice and Bob are close to a where someone tweeted." .
}
WHERE {
   WINDOW {
      EVENT ?id1 {
         ?e1 rdf:type :TwitterEvent .
         ?e1 :stream <http://streams.event-processing.org/ids/TwitterFeed#stream> .
                        ?e1 :twitterName ?someone .
                        ?e1 :location [ geo:lat ?Latitude1; geo:long ?Longitude1 ] .
         }
      SEQ
      EVENT ?id2 {
         ?e2 rdf:type :TaxiUCGeoLocation .
         ?e2 :stream <http://streams.event-processing.org/ids/TaxiUCGeoLocation#stream> .
                        ?e2 :location [ geo:lat ?Latitude2; geo:long ?Longitude2 ] .
                        ?e2 uctelco:phoneNumber ?alice .
         }
                        FILTER fn:abs(?Latitude1 - ?Latitude2) < 0.1 && fn:abs(?Longitude1 -
?Longitude2) < 0.5
      SEQ
      EVENT ?id3 {
         ?e3 rdf:type :TaxiUCGeoLocation .
         ?e3 :stream <http://streams.event-processing.org/ids/TaxiUCGeoLocation#stream> .
                        ?e3 :location [ geo:lat ?Latitude3; geo:long ?Longitude3 ] .
                        ?e3 uctelco:phoneNumber ?bob .
         }
         FILTER fn:abs(?Latitude2 - ?Latitude3) < 0.1 && fn:abs(?Longitude2 - ?Longitude3) < 0.5
                                    && ?alice != ?bob
   } ("P120M"^^xsd:duration, sliding)
}
```

Listing 5 - EP-SPARQL Query for "Contextualized Latitude" Scenario

5.3 Pachube Feed and SAN Engine Application

The purpose of this application was to show the integration of SAN engine (which is the backbone of ESR) with the M12 portal. So, the main functionalities that were tested and currently provided through the PLAY WebApp are:

1. SAN engine subscribes for specific RDF events
    - We use the Pachube Feed that involves RDF events about solar panels located in Ottersum (Netherlands). Specifically, through Pachube and M12 portal, we can receive events about these solar panels that carry information about the current energy that is produced in Watts (Pachube Feed: “my 1480 Wp solar panels”)

2. RDF events coming from M12 portal trigger SAN engine in order to traverse a simple SAN tree called “Preserve Hot Water in Heater”
3. SAN engine produces a recommendation and sends it back to M12 portal as a new event (Event topic: ESRRecom) according to the following logic:
    - We assume that there is solar water heater close to the specific solar panel. If the average value of energy production (the last 10 measurements) is less than 200 Watts and this measurement took place during day and peak hours (i.e. 12:00 am-17:00 pm) then recommend user to turn on the electrical water-heater. The system understands that there are a lot of clouds (low energy production) which means that there is not enough sunshine during peak hours and the solar water heater will not produce enough hot water to last for the rest of day. So, the only solution in order to accomplish the goal “Preserve Hot Water in Heater” is to follow the SAN engine’s recommendation and turn on the electrical water heater. In addition, SAN engine will not send a recommendation unless 10 minutes have passed from the previous one, while it needs at least 3 new events (new measurements) each time before checking the situation node of the SAN.

In the following figure you can see the graphical representation of the simple SAN tree that was used.

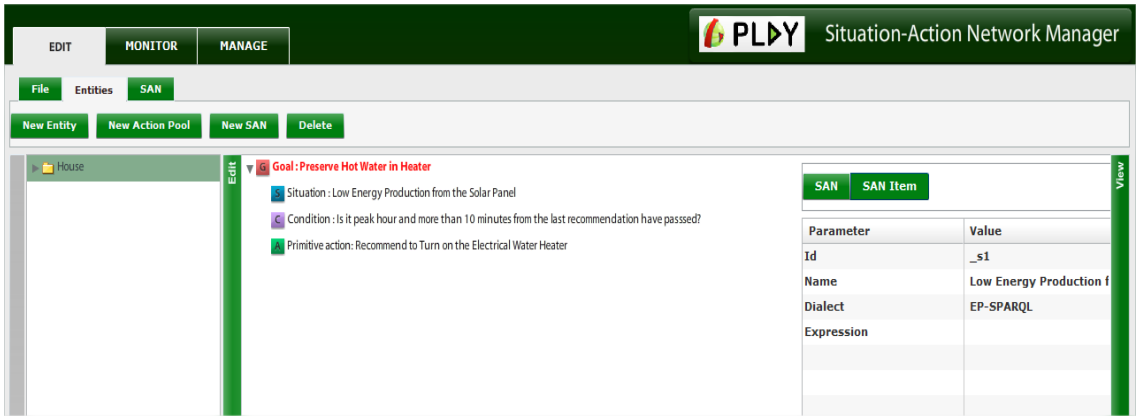

Figure 5 - SAN tree sample

5.4 Marine related ESR Application

5.4.1 Introduction

The Automatic Identification System (AIS) [16] is an automatic tracking system used on ships and by vessel traffic services (VTS) for identifying and locating vessels by electronically exchanging data with other nearby ships and AIS Base stations. AIS is intended to assist a vessel's watch standing officers and allow maritime authorities to track and monitor vessel movements. AIS does this by continuously transmitting vessels position, identity, speed and course, along with other relevant information to all other AIS equipped vessels within range. Combined with a shore stati on, this system also offers port authorities and maritime safety bodies the ability to manage maritime traffic and reduce the hazards of marine navigation.

Vessels fitted with AIS transceivers and transponders can be tracked by AIS base stations located along coast lines or, when out of range of terrestrial networks, through

[16] http://www.imo.org

a growing number of satellites fitted with special AIS receivers. The International Maritime Organization's International Convention for the Safety of Life at Sea requires AIS to be fitted aboard international voyaging ships with gross tonnage (GT) of 300 or more tons, and all passenger ships regardless of size.

The PLAY AIS adapter receives raw NMEA AIS data over internet from AISHub.net, an AIS data sharing centre, combines them and publishes to the PLAY DSB with WS-Notification two event streams in RDF/TRIG format. Aishub.net retransmits messages from over 13.000 ships and 200 base stations.

5.4.2 AIS Adapter

There are 27 different types of top level messages defined in ITU 1371-4 that can be sent by AIS transceivers. Among them the AIS adapter decodes two categories of messages which include the types of messages that contain vessel position information and the messages that contain static vessel data (such as name, type, etc.).

Every vessel equipped with an AIS transceiver sends among other the following data every 2 to 10 seconds depending on a vessel's speed while underway, and every 3 minutes while a vessel is at anchor:

- The vessel's Maritime Mobile Service Identity (MMSI) – a unique nine digits identification number.
- Navigation status – "at anchor", "under way using engine(s)", "not under command", etc.
- Rate of turn – right or left, from 0 to 720 degrees per minute
- Speed over ground – 0.1-knot (0.19 km/h) resolution from 0 to 102 knots (189 km/h)
- Longitude – to 0.0001 minutes
- Latitude – to 0.0001 minutes
- Course over ground – relative to true north to 0.1°
- True heading – 0 to 359 degrees (from a compass)

In addition, the following data are broadcast every 6 minutes:

- IMO ship identification number – a seven digit number that remains unchanged upon transfer of the ship's registration to another country
- Radio call sign – international radio call sign, up to seven characters, assigned to the vessel by its country of registry
- Name – 20 characters to represent the name of the vessel
- Type of ship/cargo
- Dimensions of ship – to nearest meter
- Draught of ship – 0.1 meter to 25.5 meters
- Destination – max. 20 characters
- ETA (estimated time of arrival) at destination – UTC month/date hour:minute

The PLAY AIS adapter combines those raw AIS messages in order to produce the following event streams:

a) **Vessel stream.** This event stream combines the vessel identification data with the position data and produces an event stream which contains both information about vessel's static data such as Name and Type and vessel data such as Latitude/Longitude, Course, Speed, Wind Speed, Wind Direction (Figure 6).

b) The vessel proximity stream. This event stream consists of events informing about the mmsi, and the distance of all nearby moving vessels within a configurable range from each vessel (Figure 7).

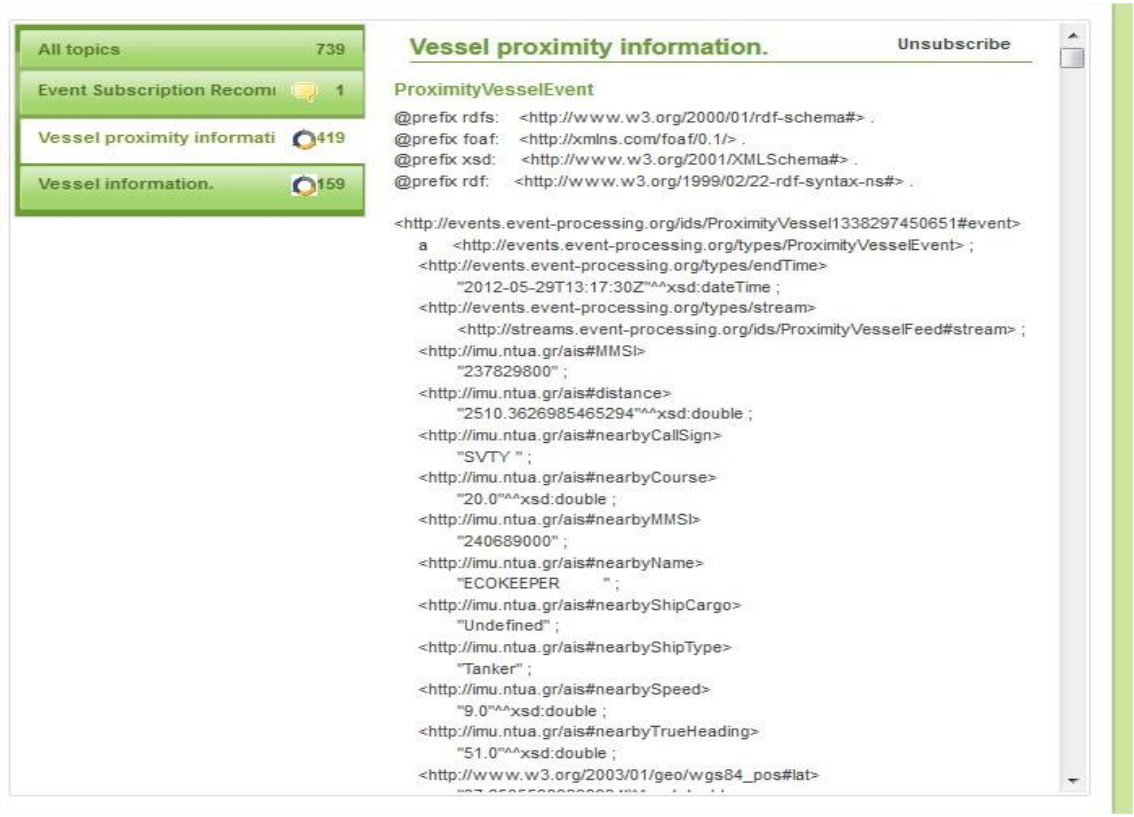

Figure 6 - Vessel Event published to the PLAY portal

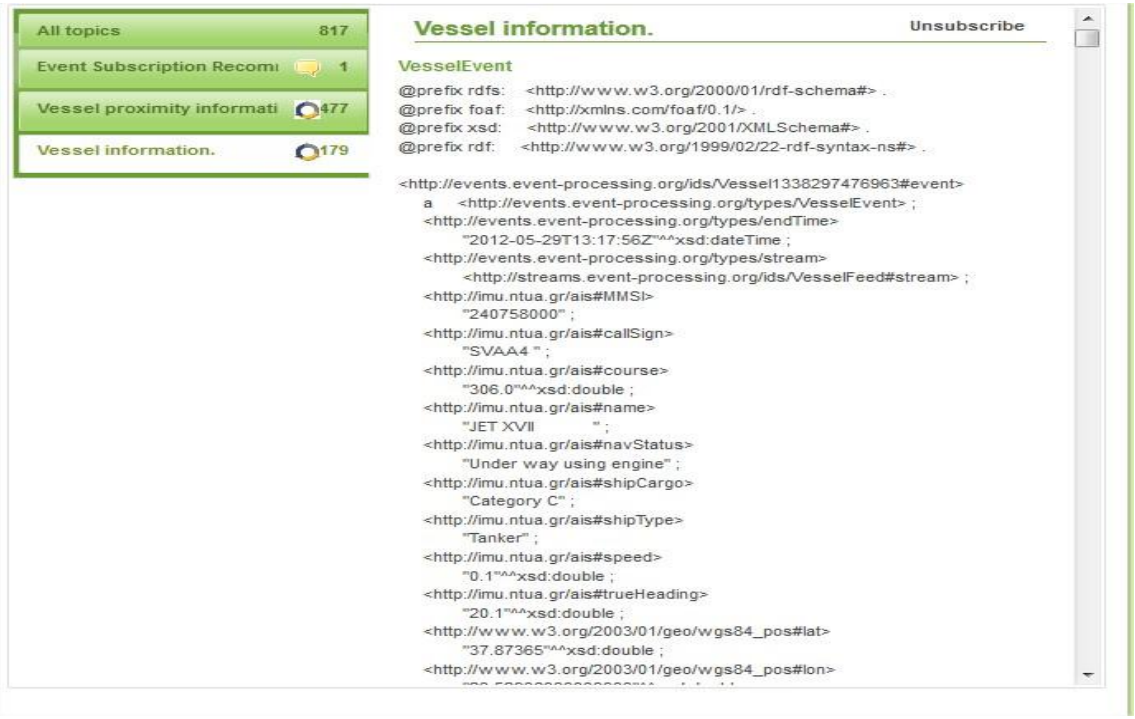

Figure 7 - Vessel Proximity Event published to the PLAY portal

5.4.3 ESR Application

The specific ESR application involves a Greek port authority that needs to enforce safety measures for high-speed vessels that are under its jurisdiction. This scenario (implemented for the China Sea during ESR testing phase) was thoroughly discussed in the deliverable D4.2. In this section we repeat only its main points. We examine two safety critical situations: i) vessels not maintaining low speed during windy conditions and ii) vessels not reducing speed when smaller boats or other high speed boats appear in proximity. We used the AISHub.net portal in order to acquire vessel related events and ESR Editor in order to design the appropriate SAN (Figure 8). Currently this ESR application is active in the PLAY portal and involves the constant processing of real-time AISHub events about vessels located at the Aegean and Ionian Seas.

In order to achieve the root goal of the "Safety in high speed crafts" SAN, ESR deploys a complex event that corresponds to the "HighSpeed Craft is Speeding" situation (i.e., subscribes to all events about vessels with speed greater than 15 knots AND Vessel type equals to High Speed Craft). The root goal leads to a parallel any complex action that spawns three goals; if one of them is achieved successfully, the root goal succeeds, too. The goal "Keep Low Speed in Windy Conditions" leads to the deployment and subscription to a new complex event that detects the situation "High Speed in Windy Conditions" (HighSpeed Craft average speed greater than 15 knots, the last 20 minutes AND Wind Velocity greater than 18 knots).

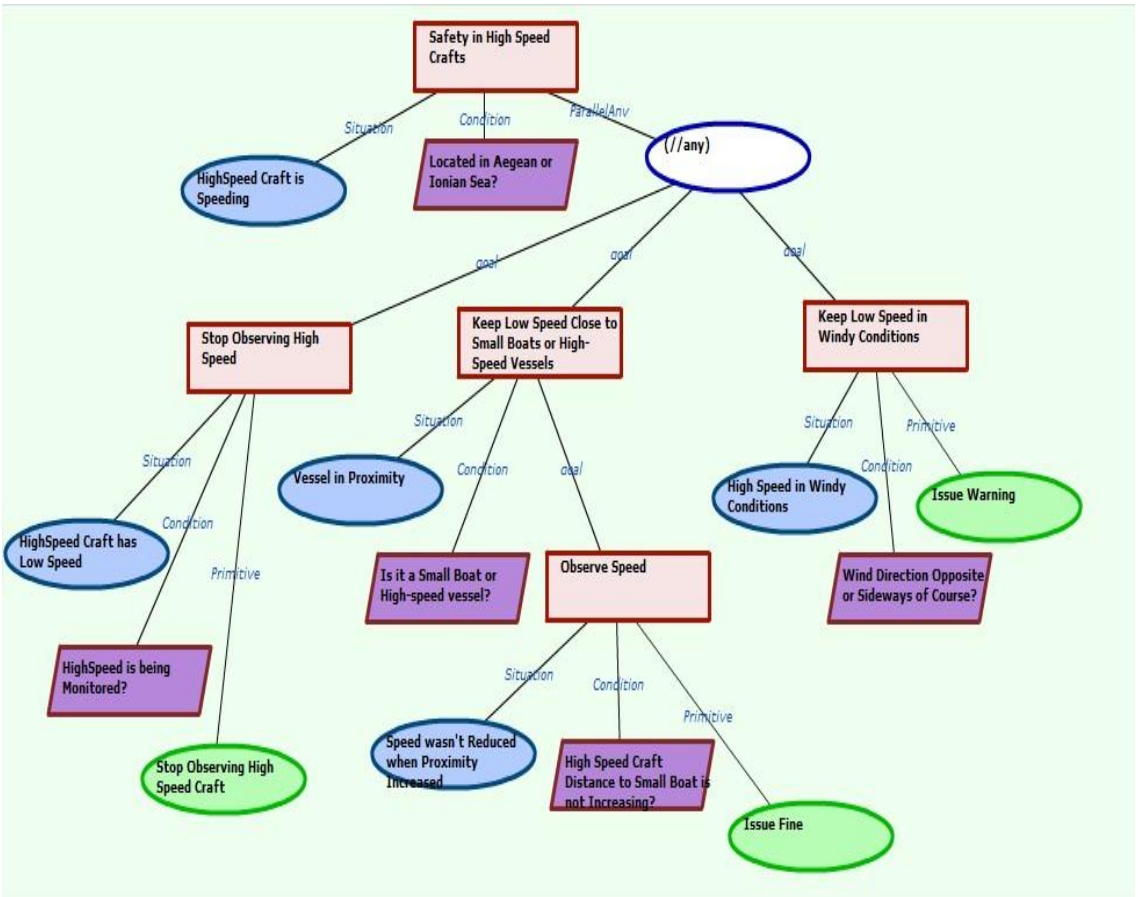

Figure 8 - Safety in high speed crafts SAN

The context node that follows, updates the craft's context (i.e. course, wind direction and velocity) and checks whether or not the wind direction is opposite or sideways of the craft's course. The specific goal is achieved when the primitive action, i.e. issue

warning, is triggered successfully. The goal “Keep low Speed Close to Small Boats or other HighSpeed Crafts” involves the deployment and subscription to a complex event that detects the situation “Vessel in Proximity”. This complex event refers to all nearby vessels to High Speed craft that are closer than 5 NM. In the context node, we examine the type of the vessel in proximity and if it is a fishing boat, pleasure or high-speed craft, yacht or sailing vessel, then we monitor the craft’s speed by traversing the respective lower level goal. This goal enables subscription to complex events in order to realize whether or not the craft’s speed has been reduced (less than 15 knots) when the distance is less than 5 NM. Since these two crafts do not deviate, ESR triggers the primitive action which is an alert for the port authority to issue a fine to the high-speed craft. The last goal refers to cease monitoring the specific craft since its speed has been reduced in less than 12 knots. Further details concerning the specific scenario can be found in the deliverable D4.2.

5.5 Management of Phone Calls

The Telecom use-case is mainly implemented through on an Android Application sending *GeolocationEvents* and *CallEvents* (see [3], [5] and [6]).

These Events are pushed from the Missed Call Manager (MCM) application running on the Smartphone to the PLAY platform thanks to the DSB component. They are classified as *Events In* from the platform point of view (considered as Event Suppliers).

Thereafter the basic Hierarchy for *Events In* is presented. Then the focus is put on *GeolocationEvent* type with a detailed description of a payload for an event instance.

5.5.1 Event Type

The different event types pushed to the platform are:

- *GeolocationEvent*: indicates regularly the position of the Caller and the Callee to the PLAY platform,
- *CallEvent* (Incoming, Outgoing): indicates a missed call to the PLAY platform with Caller ID, Callee ID, location and direction of the Call,
- *TwitterEvent* and *FacebookEvent*: indicates that a Callee has sent tweets or updates to his Facebook wall,
- *OutNetworkEvent*: indicates the cellular network status (overloaded, out of reach)
- *AckEvent*: Sent in response to a *RecomEvent*, when the user does or does not acknowledge a proposed recommendation from the PLAY platform.

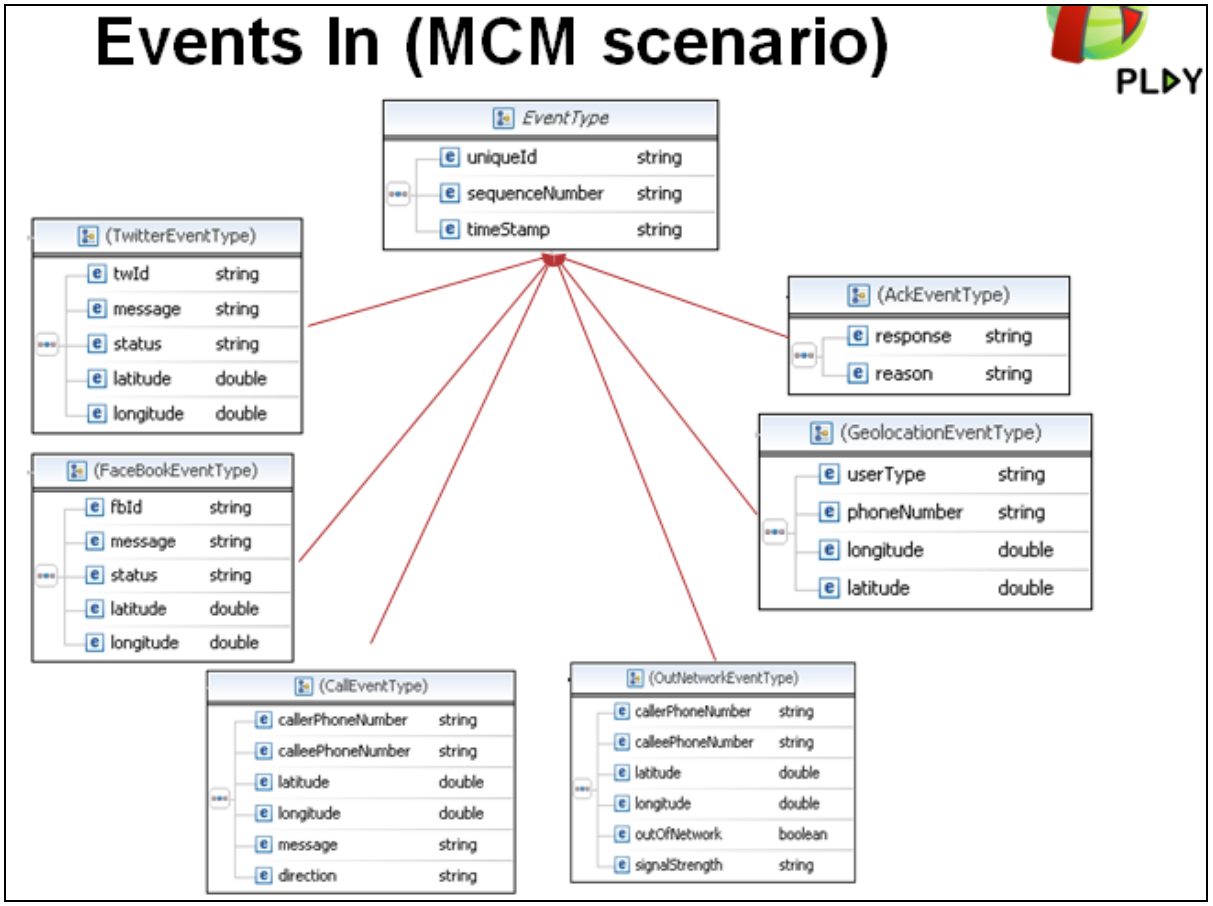

Figure 9 - Events In Hierarchy

5.5.2 Events format

Event instances are firstly generated as JAVA classes and then translated to RDF format thanks to textual templates provided by PLAY Adapters [4].

Thereafter is an example of a business message for a *GeolocationEvent* instance containing the RDF (in cyan). The RDF message is enclosed inside a WSN notification message as XML (in yellow).

```
Orange Labs Taxi UC WSN Notification:
 Geolocation events  from the Smartphone  (sent every 5s)
-------------------------------------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<wsnt:Notify xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
<wsnt:NotificationMessage>
<wsnt:Topic xmlns:s="http://streams.event-processing.org/ids/"
Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">s:TaxiUCGeoLocation
</wsnt:Topic>
<wsnt:Message>
<mt:nativeMessage xmlns:mt="http://www.event-processing.org/wsn/msgtype"
xmlns:b="http://docs.oasis-open.org/wsn/b-2"
xmlns:add="http://www.w3.org/2005/08/addressing"
mt:syntax="application/x-trig">@prefix :
&lt;http://events.event-processing.org/types/&gt; .
@prefix e:      &lt;http://events.event-processing.org/ids/&gt; .
@prefix dsb:    &lt;http://www.petalslink.org/dsb/topicsns/&gt; .
@prefix xsd:    &lt;http://www.w3.org/2001/XMLSchema#&gt; .
@prefix uctelco: &lt;http://events.event-processing.org/uc/telco/&gt; .
@prefix geo:    &lt;http://www.w3.org/2003/01/geo/wgs84_pos#&gt; .
e:3363861111740 {
e:3363861111740#event uctelco:sequenceNumber "40"^^xsd:integer ;
uctelco:uniqueId "taxiUC:3363861111740" ;
uctelco:userType "Customer" ;
uctelco:phoneNumber "33638611117" ;
# but also some other data according to event format
a :TaxiUCGeoLocation ;
:endTime "2011-12-06T18:33:36.681"^^xsd:dateTime ;
:source &lt;http://sources.event-processing.org/ids/test1#source&gt; ;
:stream &lt;http://streams.event-processing.org/ids/TaxiUCGeoLocation#stream&gt;
:location[geo:lat "43.6065"^^xsd:double ; geo:long "7.05820005"^^xsd:double] .
}
</mt:nativeMessage>
</wsnt:Message>
</wsnt:NotificationMessage>
</wsnt:Notify>
-------------------------------------------------------------------------
```

Listing 6 - Geolocation Event payload

5.5.3 Event payload

In the RDF message payload, we can extract the properties of the event instance. Below is the list of property-value for the event instance (as shown in the figure below in magenta). This *GeolocationEvent* instance has the following values:

sequenceNumber = 40
uniqueId = 3363861111740
userType = Customer
phoneNumber = 33638611117
eventType(Topic) = TaxiUCGeolocation
timeStamp = 2011-12-06T18:33:36.681
latitude = 43.6065
longitude = 7.05820005

```
Orange Labs Taxi UC WSN Notification:
 Geolocation events  from the Smartphone  (sent every 5s)
-------------------------------------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<wsnt:Notify xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
<wsnt:NotificationMessage>
<wsnt:Topic xmlns:s="http://streams.event-processing.org/ids/"
Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">s:TaxiUCGeoLocation
</wsnt:Topic>
<wsnt:Message>
<mt:nativeMessage xmlns:mt="http://www.event-processing.org/wsn/msgtype"
xmlns:b="http://docs.oasis-open.org/wsn/b-2"
xmlns:add="http://www.w3.org/2005/08/addressing"
mt:syntax="application/x-trig">@prefix :
&lt;http://events.event-processing.org/types/&gt; .
@prefix e:       &lt;http://events.event-processing.org/ids/&gt; .
@prefix dsb:     &lt;http://www.petalslink.org/dsb/topicsns/&gt; .
@prefix xsd:     &lt;http://www.w3.org/2001/XMLSchema#&gt; .
@prefix uctelco: &lt;http://events.event-processing.org/uc/telco/&gt; .
@prefix geo:     &lt;http://www.w3.org/2003/01/geo/wgs84_pos#&gt; .
e:33638611111740 {
e:33638611111740#event uctelco:sequenceNumber "40"^^xsd:integer ;
uctelco:uniqueId "taxiUC:33638611111740" ;
uctelco:userType "Customer" ;
uctelco:phoneNumber "33638611117" ;
# but also some other data according to event format
a :TaxiUCGeoLocation ;
:endTime "2011-12-06T18:33:36.681"^^xsd:dateTime ;
:source &lt;http://sources.event-processing.org/ids/test1#source&gt; ;
:stream &lt;http://streams.event-processing.org/ids/TaxiUCGeoLocation#stream&gt;
:location[geo:lat "43.6065"^^xsd:double ; geo:long "7.05820005"^^xsd:double] .
}
</mt:nativeMessage>
</wsnt:Message>
</wsnt:NotificationMessage>
</wsnt:Notify>
-------------------------------------------------------------------------
```

Listing 7 - Geolocation Event properties

5.5.4 Event properties

Finally we can define more precisely the different streams of events feeding the PLAY platform. For each event type we have properties characterizing the stream:

Stream/ Property	Stream *locationEvent*	Stream *callEvent*	Stream *Web2.0 Event Twitter/Facebook*	Stream *outNetwork Event*	Stream *ackEvent*
Source	Smartphone	Smartphone	Smartphone	Smartphone	Smartphone
Privacy	Yes	Yes	Yes	Yes	Yes
Priority	High	High	Medium	Medium	Low
Throughput	High every 5s	Low, every missed Call	Low, every Message sent	Low, every Missed Call	Low, every Missed Call

Listing 8 - Event properties

6 Dynamics of the System at Runtime

In the previous sections, we have described the modules of the platform and introduced some usage scenarios. The current section will describe how events are handled inside the PLAY platform: How to inject patterns, create event sources and how each module takes part into the global processing.

6.1 Registering a new Event Pattern in the WebApp

As mentioned several times, the PLAY platform is a distributed system of components working together. Deploying an event pattern invokes several components which must be configured automatically, e.g. DCEP to start subscribing to the necessary simple events, e.g. DSB to provide a new event topic for the results and Event Cloud to create a new cloud of storage for persisting the results.

The PLAY Web application described earlier supports several ways of editing and submitting event patterns. The tab “Pattern” opens three pattern editors (currently under development). The most flexible editor is currently the “Expert query editor” which is also the least domain-oriented.

The “Expert query editor” allows the full expressivity of EP-SPARQL including historic event queries. This editor will later support the definition of Event Level Agreements (ELAs) as soon as they are available. Figure 10 shows a screenshot of the “Expert query editor” with an example CONSTRUCT query in EP-SPARQL.

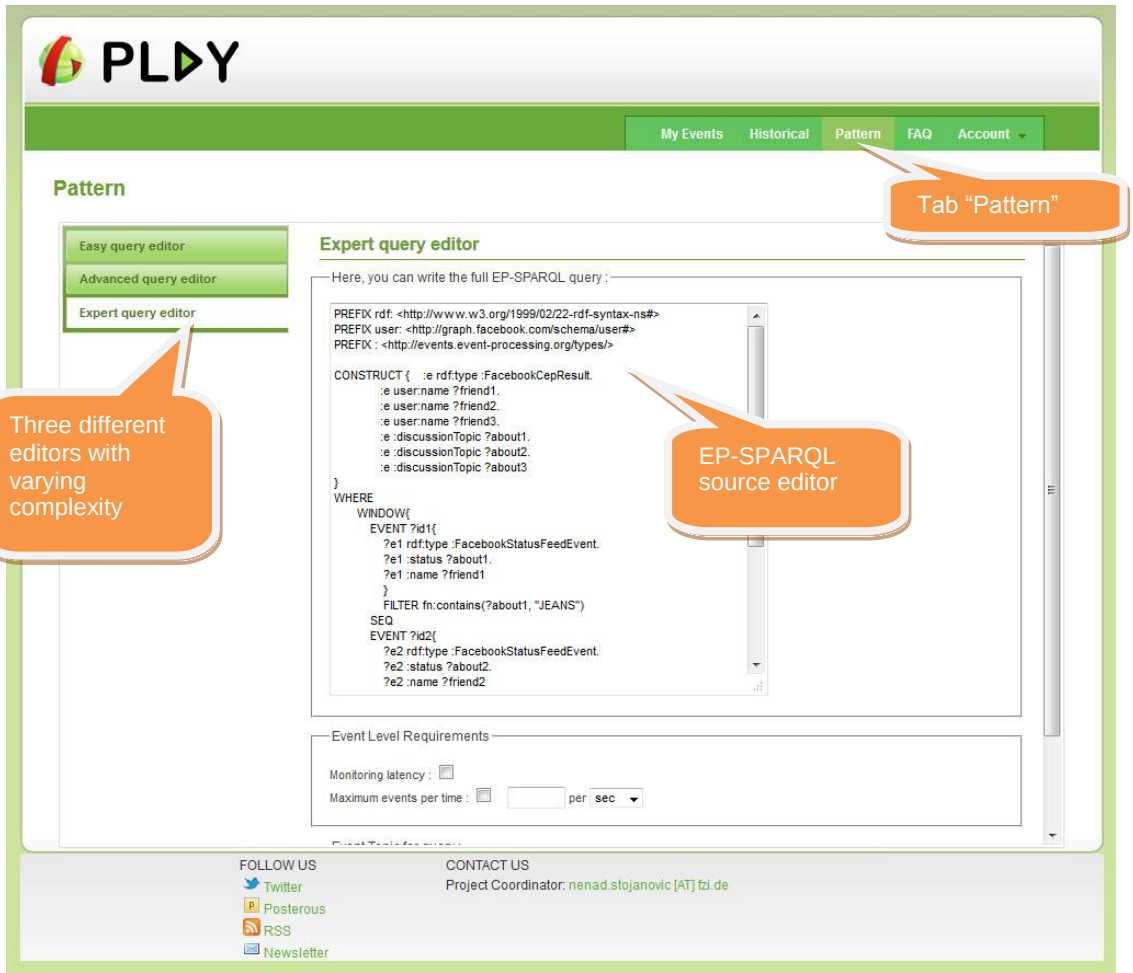

Figure 10 - Expert query editor in the PLAY WebApp

When submitting a new query, the WebApp calls the QueryDispatchApi of the QueryDispatcher component in PLAY. The call is a Web Service invocation. Then the query is decomposed in its real-time part for execution in DCEP and its historic part for execution in Event Cloud, respectively. Moreover, some important characteristics are extracted from the event pattern such as the necessary stream of simple events (DCEP must subscribe to) and the stream to hold the results (to which DCEP must publish to).

More services will be included in this process upon their completion. Figure 11 shows the design of the final process:

1. A client, such as the pattern editors from WebApp, will deploy a pattern to the PLAY governance.
2. The governance will decompose the pattern into its real-time and historic parts and extract the important stream identifiers for subscriptions.
3. The governance will then fetch the existing list of streams by getting the current topics from the Distributed Service Bus (DSB) and check if the requested topic for the results (complex events) of the new pattern already exists or not.
4. If the stream does not exist, it will be created (as a topic in DSB and as a cloud in Event Cloud).
5. If the client has requested to be subscribed to the newly create stream, this step is performed next.
6. The pattern is submitted to DCEP to start the detection. As part of the process, DCEP subscribes to the necessary streams of simple events defined in the pattern.
7. Notifications e.g. by the Facebook event adapter which happen from this point on are forwarded through the different components into DCEP.
8. A complex event is notified by DCEP and propagated to the client.

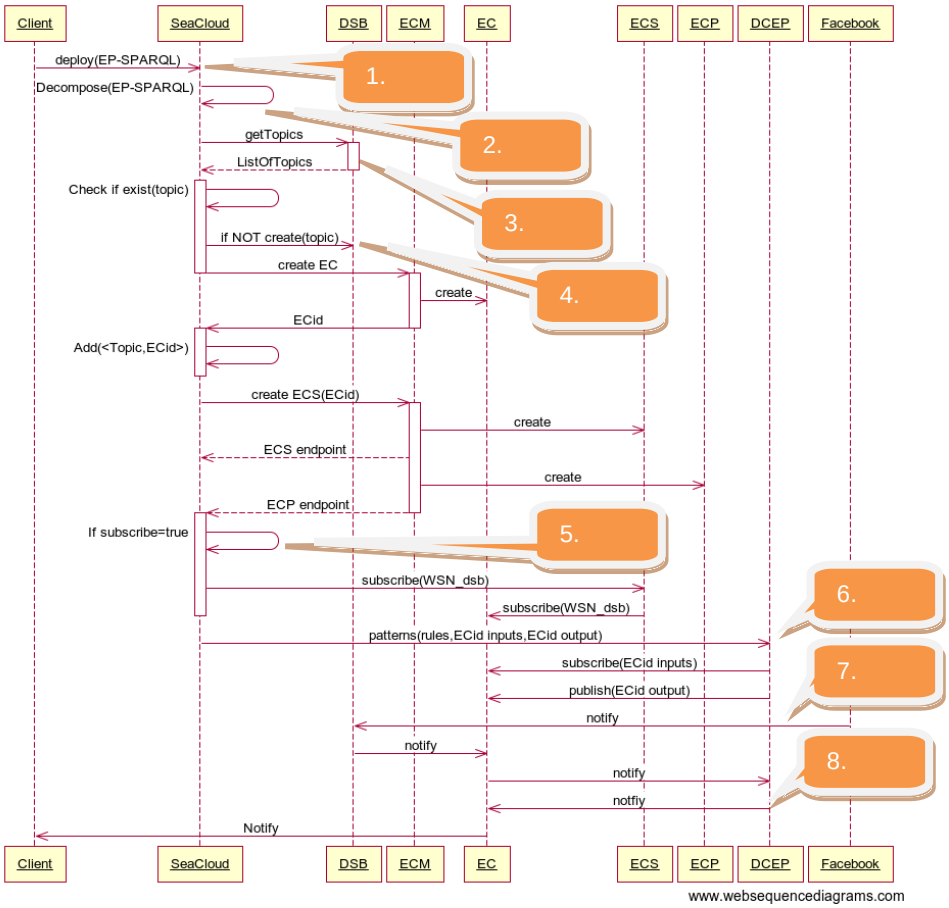

Figure 11 - Sequence Diagram of adding an Event Pattern

6.2 Adding a new Event Source to the Platform

Adding a new event source to the platform requires several steps. For most of these steps PLAY provides an SDK to build a new event adapter. For other steps existing tools can be used, e.g. to create an event schema in RDFS.

To implement an event source, an event adapter must first format events in the PLAY Event Format as described in deliverable D3.2 Section 2.3. Then the events must be published on the DSB on the correct topic.

1. There is a hierarchy of event types currently defined in an RDFS file. This file is to be superseded by the PLAY Event Marketplace.
2. The event hierarchy is automatically turned into useful Java classes which can be used after including the Maven artefact below. The classes provide getters and setters for all event attributes defined in the schema and defined on the superclasses.

There is the so-called PLAY SDK to simplify this work (except for the creation of new event types and topic).

You must add the following Maven dependency to your event source:

```
<dependency>
  <artifactId>play-eventadapter-abstractrdfsender</artifactId>
  <groupId>eu.play-project</groupId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

After the design is completed the following runtime configuration needs to be undertaken, as shown in Figure 12. The event adapter probably needs a new topic in the DSB. The events must be published to the DSB via SOAP, wrapped in a WS-Notification envelope.

1. A stream should be created to separate the new events from existing streams. This involves a new topic with the DSB and a new cloud with the Event Cloud.
2. Subscriptions are made to the new stream.
3. Any subsequent notifications are properly forwarded to the subscribed components.

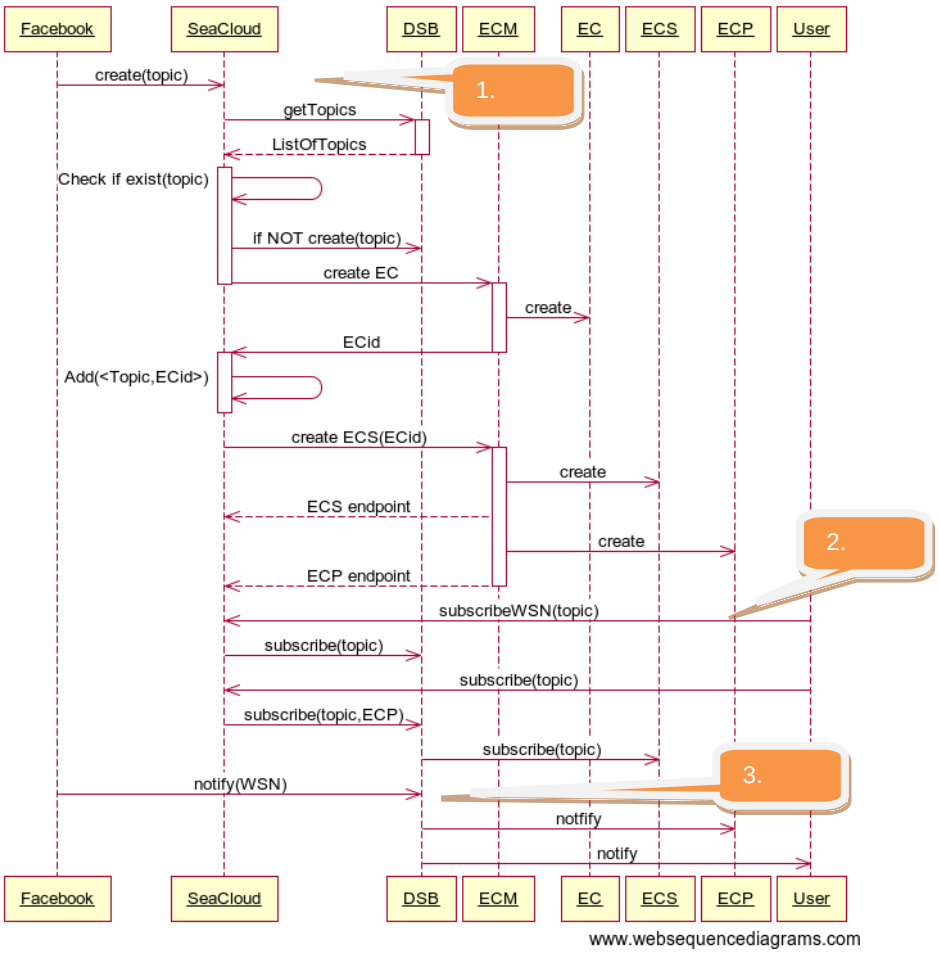

Figure 12 - Sequence Diagram of integrating a new Event Source

6.3 Adding a new Event Sink to the Platform

Creating a receiver to consume events from the platform is easier than creating a source and involves a subset of the steps described above.

There is the so-called PLAY SDK to simplify this. You must add the following Maven dependency to your event sink:

```
<dependency>
  <artifactId>play-eventadapter-abstractrdfreceiver</artifactId>
  <groupId>eu.play-project</groupId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

Your adapter must be able to receive a WS-Notification message sent by the DSB by opening some sort of HTTP endpoint. An implementation example is:

- Implement a java servlet the DSB can call to send new events.
- The incoming message payload (WSN message) must be translated into XML or string and passed to the adapter.
- The adapter decodes the message and processes it.

7 References

[1] Facebook application Web page - https://developers.facebook.com/docs/guides/canvas/

[2] D1.3 PLAY requirements analysis http://www.play-project.eu/index.php/documents/doc_download/106-play-d13-requirements-analysis-final.html

[3] Telecom use-case - PLAY Event Types http://research.petalslink.org/display/play/Event+Types

[4] PLAY Event Adaptors http://research.petalslink.org/display/play/Event+Adapter

[5] D6.1.1 Scenario of the Telecom Use-Case http://www.play-project.eu/documents/viewdownload/3-deliverables-final/24-play-d611-scenario-of-the-telecom-use-case

[6] CommonDataTaxi Sample https://svn.petalslink.org/svnroot/trunk/research/projects/play/play-usecases/play-usecase-telecom/commonDataTaxi/