Prototype of Esper-based DAQ expert system

Tomasz Bawej

October 1, 2014

Contents

1 Overview

1.1 Goal

Level **0** **A**nomaly **D**etective, commonly abbreviated as L0AD, has been implemented in an effort to evaluate Esper as a potential backbone of the CMS DAQ expert system. The project consists of a collection of EPL scripts for processing events as well as components responsible for definition and retrieval of events, registration of EPL statements and displaying the outcome of analyses. The project has never reached a mature state, hence parts of it are not well structured.

1.2 Data sources

The program receives input data in the form of events. While Esper allows several possible event representations, logically a single event is a collection of labelled data, a map. For an event to be fed into the Esper engine it has to be delivered in a predefined stream. Streams define the structure of events they can carry so that all events entering a particular stream would contain fields of the same names and types.

In L0AD, the definition and continuous population of event streams is handled by `EventsTap` objects. The project contains two implementations of taps: one for retrieving the events from a database and another for fetching them as they are published on the web. More implementations could be added by subclassing the `EventsTap` or any of its descendants.

1.3 Data input and processing

Once the event streams are defined, the engine can be instructed to detect occurrences of specific circumstances. Statement objects represent instructions for the engine and Esper provides a bidirectional conversion mechanism between a statement object and its textual EPL representation. Thus, all the queries have been saved in epl files under the `epl` directory. At runtime these files are deployed as modules by a `FileBasedEplProvider` object that relies on Esper's capabilities in that matter. `EventProcessor` class plays a role of Epser event processing engine's wrapper that facilitates certain manipulations and handles certain initialization tasks, e.g. registration of helper methods available from the EPL level.

1.4 Output

The project introduces an `EventsSink` abstract class to handle outputting the information from the Esper event processing engine. Provided implementations include `FileSink`, `SwingGui` and `ThroughputMonitor` classes, where `ThroughputMonitor` only outputs the performance-related information. It is up to the `EventsSink` implementations to provide instances of `com.espertech.esper.client.StatementAwareUpdateListener` object for statements annotated with `@Verbose` and `@Watched` annotations. `EventProcessor`'s constructor makes sure that every registered statement annotated with `@Verbose` or `@Watched` gets a listener attached so that the statement-related output can be sent to all registered `EventSink`s that provide corresponding listeners.

1.5 Helper and convenience classes

Apart from the core classes listed above, the most notable helpers include:

- `FieldTypeResolver` - a class storing (hardcoded) information about the datatypes of particular fields and facilitating the conversions.
- `Trx` - a set of convenience methods callable from EPL statements

- `HwInfo` - as above, yet related to CMS hardware
- `Settings` - a global singleton for handling the configuration values

2 Details

2.1 Configuration properties

This section aims at explaining the meaning and usage of configuration constants used in the `load.properties` file. These constants are mostly defined somewhere on the Java side, yet some of them are grouped in the `Settings` class - mostly the options reused across the whole project - while the others are introduced only by the components relying on them. This is a known inconsistency and one of many things that should be organized.

2.1.1 Proxy parameters

```
socksProxyHost=127.0.0.1
proxySet=true
socksProxyPort=1080
```

2.1.2 Flashlist database settings

Events can either be read from a database, like in the example below, or saved while being read from the dumps (as indicated in section 2.1.4). These behaviors can be toggled by the following option:
`flashlistDbMode=read`
being set either to `read` or `write`.

Actually only mySQL is fully supported as database type and only in this case the database engine setting makes sense.

```
flashlistDbType=mysql
flashlistDbEngine=myisam

   flashlistDbHost=localhost
flashlistDbUser=load
flashlistDbPass=your_password
flashlistDbName=flashlists_rest

   retrievalTimestampName=fetchstamp
flashlistDbIndexTimestamps=true
```

2.1.3 Online flashllists

```
onlineFlashlistsRoot[0]=http://srv-c2d04-19.cms:9941/urn:xdaq-application:lid=400/
onlineFlashlistsRoot[1]=http://srv-c2d04-19.cms:9942/urn:xdaq-application:lid=400/
```

2.1.4 Dumped flashlists

```
flashlistForDbDir[0]=/depot/flashlists13.11/
flashlistForDbDir[1]=/depot/flashlists13.11_2/
```

```
flashlistForDbDir[2]=/depot/flashlists13.11_3/
flashlistForDbDir[0]=/depot/flashlists13.11_4/
flashlistForDbDir[0]=/home/bawey/Desktop/flmini/flashlistsExport
flashlistForDbDir[2]=/home/bawey/Desktop/flashlists/41
```

2.1.5 EPL directories

```
eplDir[0]=epl
eplDir[1]=epl_test
```

2.1.6 Output modules

```
view[0]=ThroughputMonitor
view[1]=FileSink
view[2]=SwingGui
```

2.1.7 Events playback range

```
timerStart=1383763860000
timerEnd=1383763860800

   dateFormat=yyyy-MM-dd'T'HH:mm:ss.SSS

   outputDir=/home/bawey/load/

   flashlists=levelZeroFM_subsys;EVM;frlcontrollerLink;frlcontrollerCard;levelZeroFM_static;
   blacklist_jobcontrol=jobTable blacklist_frlcontrollerCard=myrinetProcFile,myrinetBadEventNumber,r
blacklist_levelZeroFM_subsys=FEDS blacklist_StorageManagerPerformance=activeEPs;averagingTime;bandwi
   logSinks="ch.cern.cms.load.sinks.SwingGui";
```

2.2 Connecting to database

Dumping data The very main method of the program allows switching into dumping mode:

```
public static final void main(String[] args) {
   Thread.currentThread().setName("Level 0 Anomaly Detective");
   instance = getInstance();
   if (instance.settings.getProperty(DataBaseFlashlistEventsTap.KEY\
      _DB\_MODE, "read").equalsIgnoreCase("write")) {
     MysqlDumper.main(args);
   } else {
     instance.defaultSetup();
   }
  }
```

Which in fact was an ugly way to merge the original dumper into the project. Anyway, it might still work.

HwInfo dumping During development it was possible to dump the HwInfo database into file and load it afterwards to speed the program launch up. It however required several modifications to the framework (making all the classes involved in dumping implement Serializable) that have not been introduced into it. Hence the HwInfo class still examines the settings for options specifying the HwInfo dump file to write to / read from, yet without providing these it will simply

use the remote DB. EPL annotations As mentioned above, L0AD introduces two annotations that can be used in EPL statements: @Verbose @Watched The options they take is described in the source code.

EPL statements (Please note: for the statements to be effective, the events have to be defined)

Event definitions Event definitions are dynamically fed into the program every time it starts up, based on the

Data types resolving The FieldTypeResolver class stores information about the types that particular field should be converted to. This can be specified both globally, for all fields of given name, or locally for particular event type. Unfortunately, these rules are currently hard-coded into the class. Implemented checks

3 Use cases

The project has never moved beyond the prototype phase and there is no deployment mechanism in place or whatsoever. So far it has only been launched directly from the Eclipse IDE.

3.1 Offline analysis

In order to perform offline analysis, program has to be configured to use the *Events Database*[4.4] and a *Hardware Configuration Database* connection [4.2] needs to be set up.

```
flashlistDbMode=read
flashlistDbType=mysql
flashlistDbHost=localhost
flashlistDbUser=load
flashlistDbName=flashlists_rest
```

3.2 Online analysis

Online analysis requires a slightly different approach: *Events Database* should be disabled in favor of of providing `onlineFlashlistsRoot` entries pointing to a network location to retrieve the flashlists from. Connection to the *Hardware Configuration Database*[4.2] remains a requirement, while a SOCKS proxy[4.3] configuration is also needed.

Listing 1: Sample options for fetching flashlists over the network

```
onlineFlashlistsRoot[0]=http://srv-c2d04-19.cms:9941/urn:xdaq-
    application:lid=400/
onlineFlashlistsRoot[1]=http://srv-c2d04-19.cms:9942/urn:xdaq-
    application:lid=400/
```

3.3 Dumping data for offline analyses

3.3.1 Dumping online data

Originaly the data was saved to and played back from files only. Thus, no mechanism has been implemented to dump the data directly into a database. Instead, provided that flashlists location is supplied and reachable, their contents are dumped into the folder specified by configuration option `THAT_IS_CURRENTLY_LOST_ALONG_WITH_THE_CODE_THAT_USES_IT`. Within that directory a subdirectory is creatred for each flashlist type. Flashlist rows are dumped into the files named 0, 1, 2... and so on - unique rows only, switching to the next file once the current one reaches the size of 256MB. Dumping also involves adding information about the time of fetching the flashlist - the `fetchstamp` column.

Listing 2: Sample directory structure of dumped flashlists data

```
.
+-- urn:xdaq-flashlist:BU
|      +-- 0
|      +-- 1
|      +-- 2
|      +-- 3
|      +-- 4
|      +-- 5
|         -- 6
+-- urn:xdaq-flashlist:diskInfo
|      +-- 0
|      +-- 1
|      +-- 10
|      +-- 2
|      +-- 3
|      +-- 4
|      +-- 5
|      +-- 6
|      +-- 7
|      +-- 8
|         -- 9
(...)
+-- urn:xdaq-flashlist:StorageManagerPerformance
       -- 0
```

3.3.2 Populating database

Switching `flashlistDbMode` parameter to `write` makes the `Load` class `main` method invoke the `main` method of the dumper to pass control into what used to be a separate application. Apart from

```
flashlistDbMode=write
flashlistDbType=mysql
flashlistDbHost=myDbHost
flashlistDbUser=myDbUser
flashlistDbName=myDbName
flashlistDbPass=myDbPass


flashlistForDbDir[0]=/depot/flashlists13.11/
flashlistForDbDir[1]=/depot/flashlists13.11_2/
flashlistForDbDir[2]=/depot/flashlists13.11_3/
flashlistForDbDir[0]=/depot/flashlists13.11_4/
```

The configuration above assumes each of *root* flashlist directories contains a set of subdirectories named after the flashlist they hold the values from. Listing below shows an example of such directory structure.

- Event DB connection for writing
- Flashlists dumped on the disk

3.4 Dumping online data

As mentioned above, the flashlists were firs dumped into files and only afterwards into a databases.

- Online flashlists connection

4 Configuration requirements

4.1 RCMS framework

A somewhat old version of the framework is required, in the development environment it was the revision 8055 of `https://svn.cern.ch/reps/rcms/rcms/trunk/framework`.

4.2 Hardware Configuration Database

Hardware Configuration DB connection is required for the `HwInfo` class to work. `HwInfo` provides helper methods that can be used in EPL statements to retrieve information about the hardware. In development environment the connection was setup using a port-forwarding option in `/.ssh/config`: `LocalForward 10121 cmsrac11-v:10121`, a hard-coded DB url: `jdbc:oracle:thin:@localhost:10121/cms_omds_tunnel.cern.ch` and an active ssh connection with *cmsusr*. Building the project within *Eclipse* requires the *framework* project as dependency (trunk version as of 26.06.2014).

4.3 proxy

Fetching flashlists as they are published was achieved via configuring ssh (`DynamicForward 1080` option), specifying the address to fetch the flashlists from and settings for SOCKS Proxy:

```
socksProxyHost=127.0.0.1
proxySet=true
socksProxyPort=1080
```

4.4 Events Database

The Events Database was setup during development to facilitate events playback, especially any partial playback involving arbitrary start and end timestamp. Database structure automatically mimicked the structure of published flashlists, ie. a table was created for each flashlist type and a column for each flashlist field. For convenience, all fields were stored as `VARCHAR` and one additional numeric column was added to each table: `fetchstamp` indicating the timestamp of fetching the flashlist. An extra table `fetchstamps` stores all the unique `fetchstamp` values for all other tables. Also, tables storing useful events (ie. used for analyses) have been indexed by `fetchstamp` column. This minimizes "query misses" - attempts to retrieve data for a `fetchstamp` with no corresponding events.

Password can be specified using the `flashlistDbPass` parameter. *MySQL* is the only fully supported database type with some rudimentary *MongoDB* also exists, but needs to be worked on to be useful.

4.5 Overview

4.6 Details

Until the proper, Esper-powered approach was taken, resolving EPL dependencies used to be handled by the means of inserting extra `#include` statements and determining the deployment order by hand. Some remnants of that are present in the EPL loading code or in the EPL itself and should have been removed.

4.7 Implemented checks

A summary of implemented checks along with some brief notes can be found in an xml file under the `epl/xml`