Prototype of Esper-based DAQ expert system

Tomasz Bawej (tomek.bawej@gmail.com)

October 2, 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 its structure is far from perfect.

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 `EventsSink` subclasses to implement interface methods returning instances of `com.espertech.esper.client.StatementAwareUpdateListener` for statements annotated with `@Verbose` and `@Watched`. `EventProcessor`'s constructor makes sure that every registered statement annotated with `@Verbose` or `@Watched` gets connected to compatible `EventSinks`, i.e. the ones providing appropriate listeners. This way `EventSinks` car receive statement-related updates and produce the output.

1.5 Helper and convenience classes

Apart from the core classes listed above, the project comprises numerous utility classes with the most notable being:

- `FieldTypeResolver -` a class storing (hardcoded) information about the data types of particular flashlists' fields. It also facilitates the conversions.

- `Trx` - a set of convenience methods callable from EPL statements.
- `HwInfo` - as above, but related to CMS DAQ hardware.
- `Settings` - a global singleton for handling the configuration values.
- `LoadLogCollector` - a class collecting Esper logs and dispatching them to registered `LogSinks`. It is used to display the results and the logs side-by-side in the same GUI.

2 Use cases

The project has never progressed beyond the prototype phase and there is no deployment mechanism in place or whatsoever as so far it has only been launched directly from the Eclipse IDE. Also the term *usecase* refers more to how the code could be reused, rather than to the features of a complete piece of software.

2.1 Offline analysis

In order to perform offline analysis, the project has to be configured to use the *Events Database*(see section 4.4 for details) and a *Hardware Configuration Database* connection (see section 4.2 for details) needs to be set up. An example of corresponding events database configuration is shown below.

```
flashlistDbMode=read
flashlistDbType=mysql
flashlistDbHost=localhost
flashlistDbUser=load
# assumes password-free access for user load, preferably read-only
flashlistDbName=flashlists_rest
```

2.2 Online analysis

Online analysis requires a slightly different approach: *Events Database* should be disabled in favour of of providing `onlineFlashlistsRoot` entries pointing to a network location to retrieve the flashlists from (for example see below). Connection to the *Hardware Configuration Database*(see section 4.2 for details) remains a requirement, while a SOCKS proxy (see section 4.3 for details) configuration is also needed.

```
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/
```

2.2.1 Dumping online data

Originally 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 a corresponding configuration option of the dumper project. The project can be found under the `extras/flashdumper` subdirectory and is independent from L0AD. Within the output directory a subdirectory is created 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. A sample output directory structure is depicted below:

```
.
+-- urn:xdaq-flashlist:BU
|   +-- 0
|   +-- 1
|   +-- 2
|   +-- 3
+-- urn:xdaq-flashlist:diskInfo
|   +-- 0
|   +-- 1
+-- urn:xdaq-flashlist:StorageManagerPerformance
       -- 0
```

Dumping also involves adding information about the time of fetching the flashlist - the `fetchstamp` column.

2.2.2 Populating database

Switching `flashlistDbMode` parameter to `write` makes the `Load` class `main` method invoke the `main` method of `MysqlDumper` class that used to be a part of a separate application at some point. A rudimentary `MongoDumper` class exists as well and during the early tests it was called from `MysqlDumper` (back then called something else) if MongoDB was the database type specified in the configuration. Work on MongoDB has since been abandoned and that awkward class arrangement has never been refactored. The source has however been retained in case the work with Mongo is resumed some day.
Listed below is a sample snippet of the configuration file depicting the database-pumping setup.

```
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 dumps of.

3 Configuration

This section introduces some overlap with section 2 in an attempt to list all the configuration options and elaborate on their meaning where they have not been discussed so far and a discussion is required.

3.0.3 Proxy parameters

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

3.0.4 Flashlist database settings

```
flashlistDbMode=read
```

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
```

This option indicates which column, in each table, is the time of fetching particular row. This value is later used for playback.

```
retrievalTimestampName=fetchstamp
```

This option determines whether the retrieval timestamp column should be indexed when initializing a database based on the dump files

```
flashlistDbIndexTimestamps=true
```

3.0.5 Online flashlists setting

```
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/
```

3.0.6 Dumped flashlists

The directories listed using the option below are scanned in search of the flashlist dump files that will be used to create and fill database.

```
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
```

3.0.7 EPL directories

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

3.0.8 Output modules

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

3.1 Flashlists

A list of flashlists to use.
`flashlists=levelZeroFM_subsys;EVM;frlcontrollerLink;frlcontrollerCard;levelZeroFM_static;gt_`

Blacklisting specified fields per flashlist.
`blacklist_jobcontrol=jobTable` blacklist_frlcontrollerCard=myrinetProcFile,myrinetBadEventNumber blacklist_levelZeroFM_subsys=FEDS

3.1.1 Others

Defines a period to play the data back for.
`timerStart=1383763860000`
`timerEnd=1383763860800`

Data format is used when converting timestamp strings into numeric values.
`dateFormat=yyyy-MM-dd'T'HH:mm:ss.SSS`

Output directory is the location that `FileSink` writes into.
`outputDir=/home/bawey/load/`

Classes to be registered as log sinks `logSinks="ch.cern.cms.load.sinks.SwingGui";`

4 Configuration requirements

4.1 RCMS framework

The project was confirmed to compile and run with revision 8055 of the *RCMS framework* (`https://svn.cern.ch/reps/rcms/rcms/trunk/framework`). Some later revisions make it impossible to compile the project against the 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 set up 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*.

4.3 SOCKS 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
```

An active ssh connection with *cmsusr* was needed, too.

4.4 Events Database

The Events Database was set up 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, i.e. 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. This minimizes "query misses" - attempts to retrieve data for a `fetchstamp` with no corresponding events.
Also, tables storing useful events (i.e. used for analyses) have been indexed by `fetchstamp` column. Otherwise the DB performance made the project unusable.

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 Implemented checks and further documentation

A summary of implemented checks along with some brief notes can be found in the xml files under the `epl/xml` directory.
Also, the `doc` directory of the project contains, apart from this report, some development notes on Esper and a presentation on the project.
The project is available on github: `https://github.com/bawey/load`.

4.6 Inconsistencies remaining

Given the lack of maturity of the project, this could well be the longest part of this report. However, this section includes only the issues that attracted instant attention while writing this report.

4.6.1 Resolving EPL dependencies by hand

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.

4.6.2 Configuration constants

Some of these string literals are centrally defined in the `Settings` class, while some other are added only by the classes using them.