Journal of Physics: Conference Series **664** (2015) 082008 doi:10.1088/1742-6596/664/8/082008

CMS - HLT Configuration Management System

Vincenzo Daponte
CERN, on behalf of the CMS collaboration
University of Geneva

E-mail: vincenzo.daponte@cern.ch

Andrea Bocci
CERN, on behalf of the CMS collaboration

E-mail: andrea.bocci@cern.ch

Abstract. The CMS High Level Trigger (HLT) is a collection of software algorithms that run using an optimized version of the CMS offline reconstruction software. The HLT uses Python configuration files each containing hundreds of "modules", organized in "sequences" and "paths". Each configuration usually uses an average of 2200 different modules and more than 400 independent trigger paths. The complexity of the HLT configurations and their large number require the design of a suitable data management system. The work presented here describes the solution designed to manage the considerable number of configurations developed and to assist the editing of new configurations.

1. Introduction

The CMS High Level Trigger (HLT) is implemented running a streamlined version of the CMS offline reconstruction software [1] on thousands of CPUs. The CMS software is written mostly in C++, using Python as its configuration language through an embedded CPython interpreter.
The configuration of each process is made up of hundreds of "modules", organized in "sequences" and "paths". As an example, the HLT configurations used for 2012 data taking comprised over 2200 different modules, organized in more than 400 independent trigger paths. The complexity of the HLT configurations together with the high number of configuration produced, used to cope with the changing LHC luminosity and specific detector conditions, require the design of a suitable data management system. The present work describes the designed solution to manage the large number of configurations developed and to assist the editing of new configurations. The system is required to be remotely accessible and OS-independent; moreover the system maintainability and the usability criteria are taken into account as key aspects.
To meet these requirements a three-layers architecture (Fig. 1) has been chosen to delegate to each layer a different key task. On top of the database "ConfDB" a logic manager has been introduced to handle the database operations, to keep track of the new versions of a given configuration, to perform the permissions checks and to send a configuration to the user in a suitable format for the user interface.

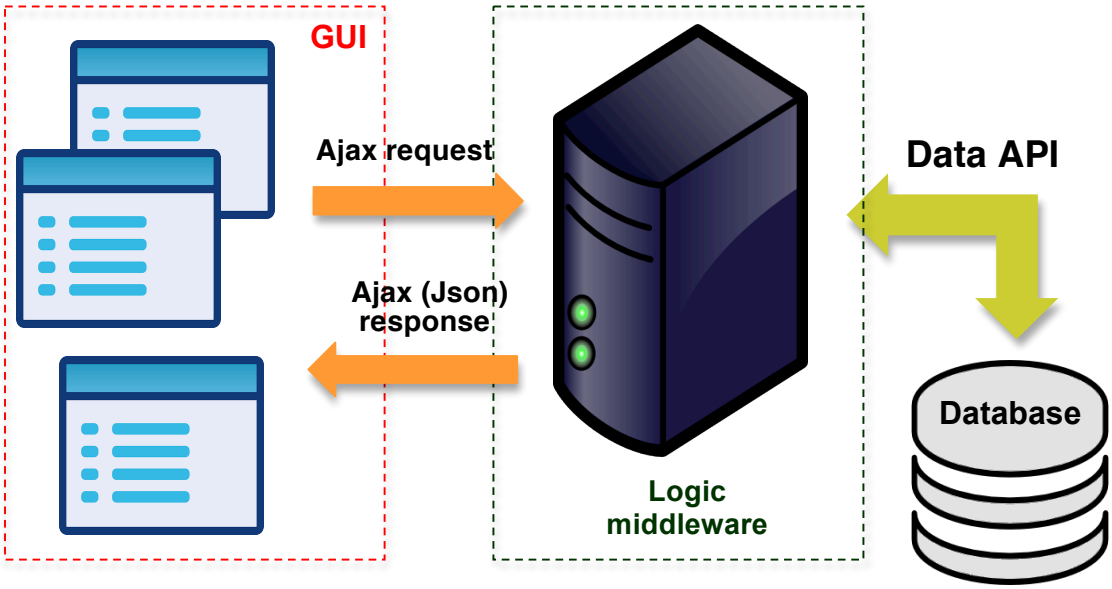

Figure 1. Three-layers architecture: On top of the storage unit (database) is built a middleware hosting the business logic, which communicate with the User interface (GUI).

The graphical user interface (GUI) will provide all the features to display, modify and manage the configurations. A web application model has been chosen for this GUI. The design was carried out first by exposing paper sketches to the end-users and on the basis of their feedback a software mock-up in HTML and JavaScript was implemented. In order not to increase the complexity of the interface, a set of JavaScript frameworks has been evaluated to cut the overhead provided by the HTML graphic design. At the end of the development process usability test will be carried out in order to measure the impact that the new GUI has on the development of configurations for the CMS-HLT.
In particular, in Section 2 the concepts of the HLT Configuration are exposed, and in section 3 the architecture of the logic middleware is outlined. Then in Section 4, the design process of the GUI is reported. Finally in section 5 the future steps are detailed.

2. Domain concept description

The starting point for the system design has been the analysis of the main concepts populating the domain of the CMS HLT configuration. A graphical representation of the entities recognized and the relationships among them is given to provide a view as complete as possible. The main entities of the HLT configuration are represented along with the relationships between them. The arrow line indicates a Generalization-Specialization relationship; while the plain line stands for a simple Association relationship. The hollow diamonds on the containing class represent the Aggregation association with a single line that connects it to the contained class; while the filled diamonds on the containing class symbolize the Composition association with lines that connect it to the contained class.

2.1. Formal definitions

In this section a formal specification of the operational context is given. This specification is formalized using natural language to define all the concepts identified in the domain. The aim of giving such definitions is to provide a unique semantic for all the objects populating the domain in order to build the relationships among them. The definitions will be listed following a hierarchical order: all the entities deriving from another one will be listed, with the appropriate indentation, below the parent entity. Further information on the relationships among the entities will be provided in the next section.

- **Process** is the top-level item of a configuration program. The Process aggregates all the configuration information for the cmsRun executable.
- **Trigger Path** is one of the main components of a **Process**, it is an ordered set of Sequences and **Modules**. It is built to identify (i.e. trigger) a specific event by the sequential execution of its components.
- **EndPath** It is a particular kind of **Trigger Path**. It is always executed after the execution of all regular **Trigger Path** is completed. It can contain several kind of Module such as **Output Module**, **EDProducer** and **EDAnalyzer**; but it can just include two particular **EDFilter**: **TriggerResultsFilter** and **HLTPrescaler**.
- **Sequence** is a component of a **Trigger Path**. It is an ordered list of **Modules**, each **Sequence** carries out a macro-task whose output can be reused in other **Trigger Path** containing the same Sequence.
- **Module** is an elementary entity and it performs a single task. There are several kind of Modules used only inside a **Trigger Path**, others only outside (still within the **Process**). The former can also be part of a **Sequence**.
    - **EDProducer** is a **Module**, it is used inside a **Trigger Path** or inside an **EndPath**. Its function is to produce new data from a given input; usually it produces higher level data with respect to the given one.
    - **EDAnalyzer** is a **Module** used inside a **Trigger Path** or inside an **EndPath**. Its main task is to analyse data previously produced. It is mainly used in the offline analyses or for monitoring purposes.
    - **EDFilter** is a **Module** used inside a **Trigger Path**. Its function is to take a decision based on the available data. EDFilters determine whether the data computed are compatible with the wanted event. If yes, the execution continues otherwise it stops.
        - **HLTFilter** is a particular kind of EDFilter used mainly during the online session.
        - **HLTPrescaler** is a apecial kind of EDFilter that can be used also inside an **EndPath**. It used to decrease the observation rate of an event, usually when this event occurs often.
        - **TriggerResultsFilter** is an EDFilter that can be used also inside an **EndPath**. Its function is to choose which results have to be stored according to a combination of trigger decisions from other Paths/Triggers.
    - **Output Module** can be used only in an **EndPath**. It specifies where the events, matching its internal selection, are saved.
    - **Source** is unique for each **Process**. It abstracts the real source where the raw data come from (i.e. the detector).
    - **ESSources** is used to include a new condition record in the configuration. The conditions can be read from an external source. All the conditions in a record have the same validity period, which is the record validity period.
    - **ESProducers** is used to import external condition values. It can write this values in existing records; it cannot create new ones.
    - **Service** is a particular class of **Module**. It provides different functionalities useful to monitor the process of data taking.
- **Parameter-Set (PSet)** It is a set of parameters. Those can be simple primitive types or structures.
- **SubProcess** is a nested **Process** inside the main one. It can have all the elements of the main one except for the **Source**, which is unique. A SubProcess can also have another nested SubProcess and so on. The execution of the nested SubProcesses is iterative.

- **Schedule** defines the execution order (i.e. the schedule) of the **Paths/Triggers** and **EndPaths** inside a **Process**.
- **Stream** represents a stream of data flowing from the detector (i.e. Point 5) to the storage unit (i.e. Tier 0).
- **Event** is the object where the outputs of the modules computation along a **Paths/Triggers** are stored.
- **Event Content** is a list of contents of an Event to keep or drop in a Stream. The contents can be dropped or kept by specifying this list through the following string format:

  keep | drop [* | Type Label Instance Process]$^{+}$ Where:

  - Type – > * | Friendly Type Name.
  - Label – > * | Module Label.
  - Instance – > * | Arbitrary String.
  - Process – > * | Process Name.

- **Dataset** is a subset of the collected data. This subset refers to a specific event or to a particular combination of events.
- **Release** indicates the CMSSW version on which the **Process** is based.

2.2. Entities representation

In this section a view on the analysed domain entities is provided. In Fig. 2 the main entities of the HLT configuration are represented. The presented diagram also shows the relationships between entities. The arrow line indicates a Generalization-Specialization relationship; while the plain line stands for a simple Association relationship.

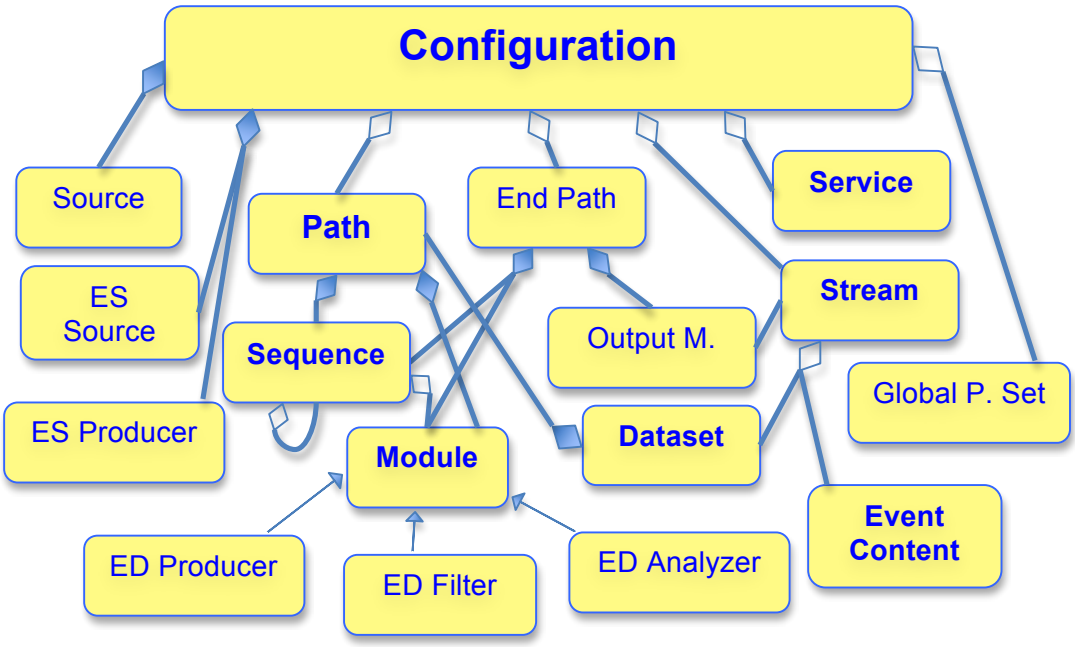

Figure 2. The diagram of the main entities of an HLT Configuration.

The main entity, which groups all the others into the final configuration, is the Process. It can be associated to:

- one or none Schedule;
- none or many Trigger Paths;
- none or many EndPaths;
- none or many Parameter-Sets;
- many Modules, in particular:
  - One and only one Source;
  - none or many Services;
  - none or many ESSources;
  - none or many ESProducers;
- and none or one SubProcess.

The second most important entity is the Trigger Path. As stated above, it includes a series of Modules used to identify a particular event. In detail it can be associated to:

- none or many Sequence;
- one or many Modules such as:
  - none or many EDFilter;
  - none or many EDProducer;
  - none or many EDAnalyzer;

Another relevant entity in the HLT configuration domain is the EndPath. It can be associated with none or many Sequences, hence with EDProducers and EDAnalyzers. It has at least one Output Modules and none or many from the two kind of EDFilter allowed in this case: TriggerResultsFilter and HLTPrescaler. There are other entities in the HLT configuration domain which are taken into account especially in the on line sessions. None or many Stream can be associated to a Process. Each Stream has a one-to-one association with an Output Module. Moreover, a Stream is associated to at least one Dataset; and each Dataset is associated to one or many Trigger Path.

3. Logic middleware

The middleware architecture is based on four components, as shown in Fig. 3, each addressing a specific task, those components are the Ajax API, the Logic module, the Serialization module and the Data API.

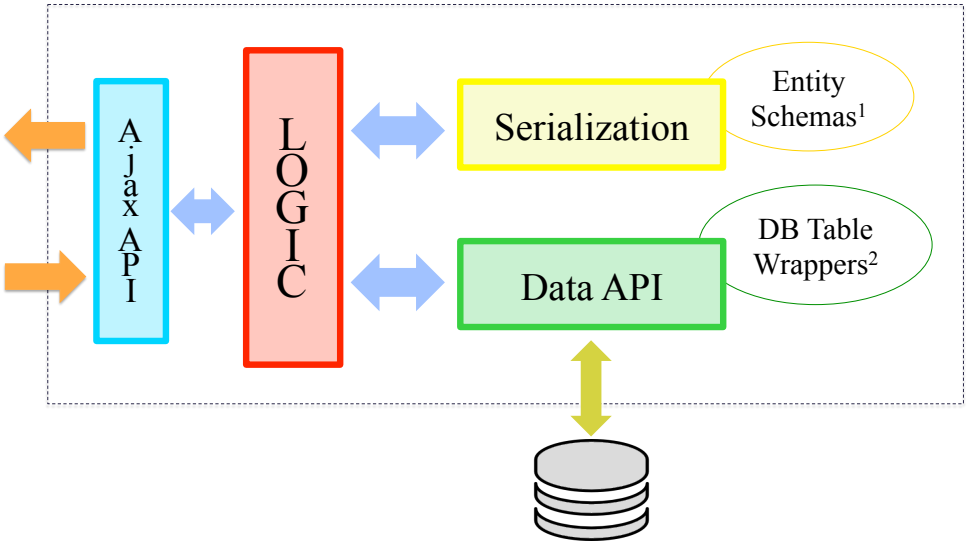

Figure 3. The structure of the Logic middleware.

3.1. AJAX API

The AJAX API provides the interface used by the GUI to access and manage the contents. This module lists all the exposed methods that can be invoked by the client application through Ajax requests. Each method provides access different contents by triggering the corresponding algorithm in the Logic layer.

3.2. Logic layer

The Logic layer implements the operations required to build the contents requested by the client and perform the required task. The method invoked first checks for the inputs to be consistent and then retrieves the necessary data through the Data API. With the collected data, the algorithm invoked builds the content requested by the client and serializes it through the Serialization module to send the response in a suitable JSON format.

3.3. Data API

The Data API provides to the logic layer a suitable interface to the database, wrapping any low level operation (i.e. database query). This module is based on two main components: the Entity schema and the Data API itself. The Entity schema abstracts the tables present in the Database at middleware level, providing a direct way to retrieve information from this data source. The Data API implements the query used to retrieve the requested contents from the database.

3.4. Serialization layer

The Serialization layer ensures the data retrieved by the Logic layer is built according to the standard used in the GUI. The Logic layer invokes this module once the content requested by client is built to serialize it into JSON format. This operation is performed relying on another schema that abstracts the content entity to the client application level; in particular the data serialized are formatted in a suitable way for being consumed by the client application.

4. GUI design

The GUI design was carried out first by exposing paper sketches, as in Fig. 4, to the end-users and based on their feedback a software mock-up was implemented.

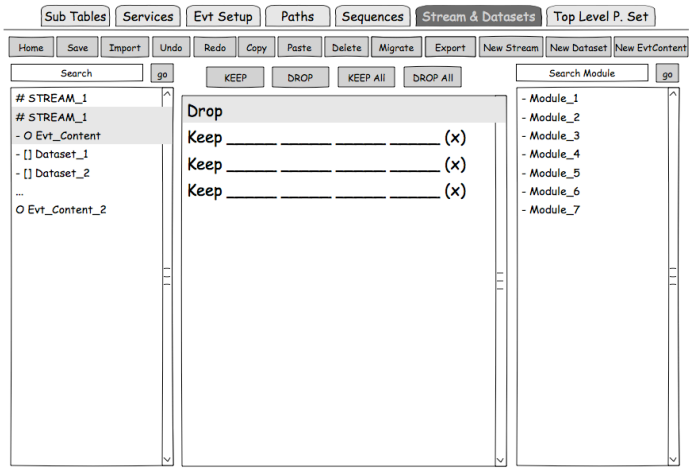

Figure 4. The sketch of a client application view.

The need to provide customized features for each main entity in the Configuration domain led us to choose the Model-View-View Model [2] (MVVM) design pattern for the GUI web application.

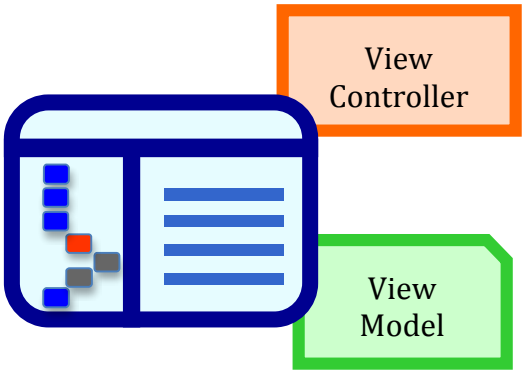

Figure 5. The Model-View-View Model schema.

In this pattern, as shown if fig. 5, each view is provided with its own controller and the instances of the domain model concerning that view. At the end of the development process, a usability test will be carried out in order to measure the impact that the new GUI has on the development of configurations for the CMS HLT.

5. Future developments

The solution exposed is meant to provide high decoupling and high flexibility among the layers. These features can be exploited in case of database schema changes as well as in case of GUI updates. In both cases just minor changes to the Data API and to the Serialization layer respectively would be required, avoiding extensive modification to the rest of the application. The addition or the substitution of a data source can be easily handled by designing a suitable Data API leaving the other components unchanged.

Additional features to be introduced are the parser module to obtain the python version of a desired configuration, this is the format used to run the CMSSW instances. The editing capabilities, already foreseen in the previous design phase are also to be implemented.

References

[1] CMS Collaboration, “CMS Physics Technical Design Report Vol I: Detector Performance and Software”, CERN/LHCC 2006-001.

[2] Li, Liu. "Analysis and Application of MVVM Pattern." Microcomputer Applications 12 (2012): 019.