Feicim: A browser for data and algorithms

Zsolt I. Lázár
School of Physics, University College Dublin, Dublin 4, Ireland, and
Faculty of Physics, Babeș-Bolyai University, 400084 Cluj-Napoca, Romania.
E-mail: zlazar@phys.ubbcluj.ro

Tahar Kechadi
School of Computer Science and Informatics, University College Dublin, Dublin 4, Ireland.
E-mail: tahar.kechadi@ucd.ie

Ronan McNulty
School of Physics, University College Dublin, Dublin 4, Ireland.
E-mail: ronan.mcnulty@ucd.ie

Abstract. As programming and programming environments become increasingly complex, more effort must be invested in presenting the user with a simple yet comprehensive interface. Feicim is a tool that unifies the representation of data and algorithms. It provides resource discovery of data-files, data-content and algorithm implementation through an intuitive graphical user interface. It allows local or remote data stored on Grid type platforms to be accessed by the users, the viewing and creation of user-defined or collaboration-defined algorithms, the implementation of algorithms, and the production of output data-files and/or histograms. An application of Feicim is illustrated using the LHCb data [1]. It provides a graphical view of the Gaudi architecture, LHCb event data model, and interfaces to the file catalogue. Feicim is particularly suited to such frameworks as Gaudi which consider algorithms as objects [2]. Instant viewing of any LHCb data will be of particular value in the commissioning of the detector and for quickly familiarizing newcomers to the data and software environment.

1. Introduction
Feicim, from the Gaelic word meaning “I see”, is a browser for data, algorithms, and jobs. It’s aim is to provide a simple graphical interface that allows the user discover and investigate data in an intuitive fashion.

The goal of the physicist is to make physics measurements using the data collected in an experiment. This requires finding and accessing the data, extracting relevant information, interpreting it, and filtering it to produce the physics quantities of interest. In particle physics, the process of understanding the data requires several passes through the data at different layers of granularity, from detector level quantities in the commissioning stage of the detector, through the creation and evaluation of distributions of physics quantities central to the measurement, to later stages of an analysis which require cross-checks on control samples to demonstrate that systematic uncertainties are understood.
However, as programming environments become ever larger and more complex, access to the information of interest becomes more difficult. A high level of technical computing expertise is often required to interact with the data. This barrier to information can lead to an inefficient use of resources on the experiment. Newcomers, particularly students undertaking a Ph.D, often spend 6-12 months familiarising themselves with the code. Even established physicists, with significant detector or physics experience may not get to grips with the data first-hand, but rely on other members of the experiment to make the histograms which then they can interpret. Closing the gap between the technicalities of data analysis and the appreciation of the physics within the data is clearly in everybody’s interests.

Feicim attempts to provide an intuitive interface for browsing

- The location of data-files (local or on the Grid), and the data content of those files.
- The algorithms available, their configuration, and generation
- The creation, submission and monitoring of analysis jobs running locally or on the Grid.

In this paper we describe the interface presented to the user to allow them interact with the data (section 2) and the design and implementation of Feicim that makes this possible (section 3). What the user sees is quite general and could be applied in any large-scale scientific setting. The particular examples shown are relevant to the LHCb experiment which uses the Gaudi software architecture [2].

2. The User Interface

Because the target user of Feicim is someone without extensive computing knowledge, it is intended that Feicim appears as simple and intuitive as possible, so that it can be operated almost without reading any documentation, with most features discovered by the user while investigating the data. It uses the same style of graphical explorer as found on most operating systems.

2.1. Browser for data

The first feature of Feicim is the ability to browse the data. This will be of particular use during the commissioning phase of the experiment, or the early stages of a physics analysis, where it is important to be able to quickly view variables of interest, or select a subset of variables for further analysis.

Traditionally this is done by consulting a database to find the logical filenames of the datasets of interest. The usual starting point for most physics applications are datasets referred to as “data summary tapes” (DSTs). Knowledge of the DST structure and the coding framework is required in order to write a small routine to access the variables of interest and to output them as ascii files, ROOT files [3], or as a micro-DST containing a subset of the full information [4]. Feicim removes the necessity of understanding and writing code within the coding framework and provides an intuitive interface that allows self-discovery of the information of interest.

Data-file discovery is achieved through a browser which presents the local filesystem and the content of the LHCb bookkeeping database in a similar fashion [5]. Feicim offers a tree-like representation of the file catalogue as shown in Figure 1. The data is logically organised into folders which correspond to the fields under which the data is organised in the file catalogue. The user navigates through these folders to find the DSTs of interest which can then be selected for further viewing. Currently Feicim will copy these DSTs locally in order to examine their content, but it is envisaged that this examination could be performed at the remote location.

The navigation through the data follows in a similar fashion. Most generally, particle physics data can be described as a graph with various inter-relationships between the data. However, from the physicist viewpoint, it is most natural to think in terms of a tree. The basic unit of classification is a physics event to which all data occurring in that collision belongs. Within
an event, data can be classified under various headings or branches e.g. detector information, tracking information, vertex information etc. Each of these branches has further subdivisions: detector information might be further subdivided into branches corresponding to each of the sub-detectors, which each in turn might have 'hit' information on what was detected in that sub-detector. However, though this grouping of information is natural to the physicist, the data is not a pure tree. An additional complication comes since information contained within one branch may be related to information on another branch: e.g. tracking information, stored in the track branch, may refer to individual detector hits which would be fully described in the detector branch. These inter-relationships between branches however occur in a well defined fashion and usually link to the base of another branch from which further information can be found. Thus the graphical model we have adopted to represent the data for the physicist is a tree, with any relationship between branches represented as a link.

This can be seen in Figure 2 where the event structure is represented by a tree of connected folders within which the data resides. It is very natural for the physicist to browse through such a structure and discover the data. The links are represented by an arrow symbol showing the branch to which they relate. Clicking on this symbol will allow a further expansion of the tree,
which will be a copy of the information that is stored at the referred position. The leaves at the lowest level of the tree correspond to scalar variables which can be selected. This information can be printed and plotted allowing a quick appreciation of the quantities (see Figure 3). Simple filters can also be applied. Finally the data can be written to file: either a ROOT file for further isolated investigation using the graphical power and analysis tools of ROOT, or a micro-DST - a smaller version of the original data structure which can be understood by, and access the tools and algorithms of, the experiment code environment.

Figure 3. Screen-shot of Feicim browser displaying data in both tabular form and as a histogram.

2.2. Browser for algorithms
The second feature of Feicim is the ability to browse and configure algorithms for the data. Given the data described above, the user could, in principle, dump whatever is of interest to file and then write their own algorithms or analysis code from scratch. However, quite apart from the needless duplication of code that would result, much effort would be spent validating this new code. Within the code framework of a particle physics experiment, algorithms already exist to perform the most common tasks. Furthermore, within a large collaboration, the publication of a user-created algorithm allows for its validation and re-use by others. The problem for other users therefore resides in the discovery of which algorithms exist, and their simple application to the data.

Feicim adopts a similar approach to algorithm discovery, representing them as icons within folders (see Figure 4) which correspond to the classes in which they reside. Clicking on an icon links to documentation on the algorithm. To configure the algorithms, they are dragged to the editing workspace where they are represented as a box containing tabs which correspond to the inputs and outputs of the algorithm. Clicking on the algorithm box allows parameters for the algorithm to be configured. Clicking on a tab shows the type of data that must be presented to the algorithm.

2.3. Browser for the creation, submission and monitoring of analysis jobs
The third feature of Feicim is to combine selected algorithms and data (as discovered above) to create and submit a user-analysis job. The analysis job is created by combining algorithms on the editing workspace linking inputs and outputs of the individual algorithms and thus defining the data-flow (see Figure 5). Type-checking ensures that the correct data is presented to each
algorithm. If the data selected is local, the job can run locally. More usually the job is submitted to the Grid (using Ganga [6] and DIRAC [7] for LHCb), with Feicim able to view and track their progress.

3. The Design and Implementation of Feicim
The LHCb software is written in C++ and uses the Gaudi framework and a number of external packages [2]. Feicim is a relatively thin layer on the top of LHCb and employs some additional third party and open software. The architecture of Feicim is shown in Figure 6.

3.1. Python
For the most part Feicim is implemented in Python 2.4. This is a natural choice due to the basic requirement of high-level user interfaces and more importantly, because of the existing Python
interface for Gaudi [8]. The interactive features of the language provide the user with a console for controlling Feicim while its dynamic data type system allows for on-the-fly generation of custom made Gaudi algorithms. The down side of using Python is the mandatory loss of control compared to statically typed languages and the fact that GaudiPython does not cover in detail the features of the underlying C++ framework.

3.2. User interface
User control is possible through two user interfaces. The primary one is graphical (GUI) while equivalent services are also available via the Python console that Feicim is started from. In a non-graphical environment, e.g. through a remote shell, the user can still have full control over their Feicim session. The GUI is written using Java for two reasons. Firstly, it allows the system to be separated into a light front-end and a remote back-end giving the possibility of installing and running the front-end on different platforms (e.g. through a Web-Browser). The intrinsic support of the language for different platforms and the wide palette of technologies for remote communication, such as Remote Method Invocation (RMI) and Remote Procedure Call (RPC), make Java the preferred technology for the interface. Secondly, the ability to create graphical elements is easily provided by JGraph, a popular library for drawing and interactive manipulation of graphs [9]; no Python module of similar capabilities is available. Bridging the two languages is done via a small package named JPype [10], which is responsible for starting up and shutting down of the Java Virtual Machine (JVM) and the overlaying of dataflow between the Python and JVM process.

3.3. Communication with LHCb Software
Feicim and LHCb software are interfaced via Bender [11] a Python based physics analysis environment relying on GaudiPython and LoKi [12], a C++ toolkit for physics analysis. High-level interfaces to I/O operations on DST files and the handling of the transient event store (TES) are handled by Bender. The only times that Feicim communicates with LHCb is when reading and writing actual data. For the most part, it works with textual meta-data describing the structure of the DST and that of the Gaudi objects. The advantage of meta-data over
Dynamic discovery of the data is the former’s ability to provide a complete description of the important aspects. Dynamic discovery can only be used on a particular instance of the data with properties that may or may not be generalized to the whole dataset of interest. Besides, Gaudi’s introspection capabilities are obscured by the software layers between Feicim and LHCb. In searching, browsing and configuring algorithms, meta-data is especially useful though it relies on the development of a LHCb standard for documenting the content of data files and algorithms.

All data and the associated meta-data has to be fetched from remote servers that are localized by a name server. Feicim is designed to work with information about various types of resources: static data, meta-data, software components, processes, hardware; all accessible from within a distributed environment. Therefore introducing a global naming system for uniquely identifying these resources is inevitable. The current resource space in HEP is only partially covered by unique names and it is done without the intent of truly global identification. One example is the logical file names used for data sets. Feicim assumes the existence of a global naming scheme and localizes resources using a stub name server.

### 3.4. Data representation

When referring to the content of a DST we mean its in-memory object structure as laid out in the TES during runtime (Figure 2). This view is basically the representation of the type tree for the objects in the TES. Parent-child links stand for object composition, e.g. a Particle has a Mother. At implementation level these relationships consist in non-callable references and methods returning the composing object. Consequently, parent-child links assume a direct association between the two objects while siblings can be considered independent. This aspect is essential for data reduction. The lowest level, the leaves of the tree, are scalar objects.

In addition to the composition information there are two extra ingredients in the Feicim representation:

- **entity relationship** between parent and child is specified by the usual $11, ln, n1, nm$ notation. This information is needed not only when performing selections on the data but it is also relevant for the user. For instance, $ln$ and $nm$ indicates that the parent objects each have an associated array of children of the same type.

- **links** represent the composition relationship between a parent and a child whose primary location is at a different location in the tree. They work similarly to symbolic links in the UNIX filesystem.

Following links the tree can be expanded indefinitely. On the other hand, while looping along a closed circuit of symbolic links in a filesystem does not have any benefits for the user the situation is very different in Feicim when it comes to reducing the data through a selection.

### 3.5. Data reduction

Depending on whether the result of the selection is wanted in tabular form (e.g. as a ROOT ntuple) or as a micro-DST, the meaning of the boxes and the data reduction mechanism are slightly different. If the final selection is wanted in tabular form, the selected quantities of interest, are stored in a MySQL database. This allows for the immediate viewing of data (as a printout or as histogram) as well as the easy application of filters to the data which is achieved through SQL statements. Every column will contain values for one of the selected quantities. Quantities that are independent, that is they are not in an ancestor-descendant relationship, can produce multiple rows equivalent to a direct product between two sets. For example, if 

```
/Event/MC/Particles/Pid
```

and

```
/Event/MC/Particles/EndVertices/Position/X
```

are selected this is equivalent to the query: "return for all events all the corresponding particles’ Pid and for all these particles the x-coordinates of the associated vertices". The ntuple is rendered as a paged table with all columns displayable as histograms (see Figure 3). The whole ntuple can be
saved as a ROOT file. Loading the ntuples and saving them to a ROOT file is done by ROOT controlled through PyROOT. When not through ROOT, the database server is accessed via the MySQLdb package, a Python interface to the MySQL C API [13].

When the result is wanted as a micro-DST file, the selected boxes indicate the line of dependence along which the objects are to be retained. The data is not cached in the MySQL database, and the objects are directly saved to a micro-DST file using the existing functionality of Gaudi.

3.6. MySQL

The visual specification of the selection is a well-defined procedure that for a given input - the DST files - produces a well-defined output - micro-DST or ntuple. As such it behaves as a language. The role of SQL is to provide an intermediate textual specification of the selection. The set of paths chosen by the user are converted into SQL queries complemented with a WHERE clause with the filtering conditions specified manually in the designated field. The conversion is relatively simple but its usage requires storing the content of the DST in the database. Using MySQL is a convenient temporary solution as it is an OpenSource, full-fledged, well supported, efficient database server that is part of every Linux distribution. However, since the DST format was optimized for efficiently storing HEP data the conversion to MySQL is expensive in terms of execution time and memory usage. It is in principle possible to develop a native module that can directly interpret the visual input and perform the extraction and selection of the data. Other options such as SQLite are considered for the future [14].

4. Conclusion and Further Work

This paper identifies the essential ingredients to a software solution that would substantially improve on the learning curve of newcomers to particle physics and enhance the productivity of researchers involved in data analysis. These elements are specified by a list of requirements and GUI design recommendations. A method for visualizing the content of data files and interactively specifying the selection of a subset of the data has been demonstrated by the current version of Feicim. We have concluded that in order to maximize the services offered by Feicim, all particle physics data and software requires associated meta-data uniquely identified within a global naming scheme.

The algorithm browser and job composition part of Feicim is under development. The major challenge here is with the specification of the job options which describe how the algorithms will function. Recent work by the ATLAS collaboration on a python module for configuring algorithms suggests one way to proceed.

Finally it is noteworthy from Figure 6 that the structure of Feicim is only loosely tied to the LHCb framework through the Bender interface. Thus with an appropriately designed interface and meta-data describing the data and algorithms, Feicim provides a framework for distributed data analysis that could be used in a wide variety of scientific disciplines.
Acknowledgments
This work is supported by the Marie-Curie Grant no. MTKD-CT-2004-003134. The contribution of Zoltán Máthé to the code, supported by the National University Research Council of Romania through grant no. 27687/14.03.2005, is gratefully acknowledged.

References
[1] LHCb Computing homepage, lhcb-comp.web.cern.ch/lhcb-comp/
[2] Cattaneo M et al 2001, Status of the GAUDI event-processing framework Proc. of Int. Conf. on Computing in High Energy and Nuclear Physics (Beijing).
Barand G et al 2000, GAUDI - A software architecture and framework for building LHCb data processing applications Proc. of Int. Conf. on Computing in High Energy and Nuclear Physics (Padova)
[3] ROOT homepage, root.cern.ch/
[4] micro-DST TWiki pages, https://twiki.cern.ch/twiki/bin/view/LHCb/MicroDST
[5] LHCb Bookkeeping database, lhcbbk.cern.ch/BkkWeb/Bkk/welcome.htm
[6] Harrison K, et al., Ganga: a Grid User Interface Proc. of XV Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP'06) (Mumbai) ed Banerjee S (Macmillan India)
Maier, A. et al., Ganga - a job management and optimising tool, Proc. of XVI Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP’07) Victoria, Canada,
Elmsheuser, J. et al., Distributed Analysis using GANGA on the EGEE/LCG infrastructure, Proc. of XVI Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP’07) Victoria, Canada.
[7] Tsaregorodtsev A. et al., DIRAC: the LHCb data production and distributed analysis system Proc. of XV Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP’06) (Mumbai) ed Banerjee S (Macmillan India)
Paterson S., LHCb distributed data analysis on the computing grid, Ph.D. Thesis, University of Glasgow (CERN-THESIS-2006-053), 2006.
[8] GaudiPython TWiki pages, https://twiki.cern.ch/twiki/bin/view/LHCb/GaudiPython
[9] JGraph homepage, www.jgraph.com/
[10] JPype homepage, jpype.sourceforge.net/
[11] Bender homepage, lhcb-release-area.web.cern.ch/LHCb-release-area/D0C/bender/
[12] LoKi homepage, lhcb-comp.web.cern.ch/lhcb%20comp/Analysis/Loki/
[13] MySQLdb homepage, https://sourceforge.net/projects/mysql-python
[14] SQLite homepage, www.sqlite.org/