Rheinisch-Westfaelische Technische
Hochschule Aachen
Fakultät für Mathematik, Informatik und Naturwissenschaften

Bachelorarbeit Physik

Development of the VISPA@WEB Program and Measurement of the Dijet Mass with CMS

von Matthias Komm

20.07.2010

Gutachter
Professor Dr. Martin Erdmann
III. Physikalischen Institut A
RWTH Aachen

Betreuer
Dipl.-Phys. Gero Müller
III. Physikalischen Institut A
RWTH Aachen

Matthias Komm:
Development of the VISPA@WEB Program and Measurement of the Dijet Mass with CMS

Bachelorarbeit Physik
Rheinisch-Westfaelische Technische Hochschule Aachen

Bearbeitungszeitraum: 20. April 2010 - 20. Juli 2010

Contents

1 Introduction

1.1 Physics analysis

In high energy and astroparticle physics physicists are often confronted with a large amount of data to deal with. The performed analysis can become very complex and this is where the common term 'physics analysis' appears. For an introductory example, I present in figure 1.1 the Feyman diagram of the simple reaction $e^+e^- \rightarrow Z^0 \rightarrow \mu^+\mu^-$ which can take place at the interaction point in a particle accelerator.

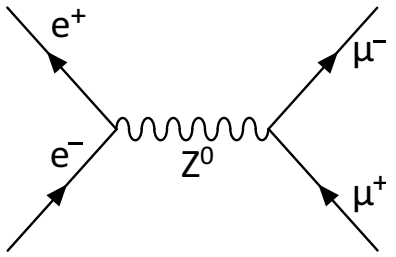

Figure 1.1: Feyman graph: Z^0

In a particle detector the Z^0 is not measured due to a decay width of $\Gamma_{Z^0} = 2.495$ GeV[14]. Therefore the collected data includes only information about the final state particles, in this case the muons[1]. Special computer software in interaction with the electronics of the detector can determine the four-vectors p_μ of the muons. For more details how this can be done see Ref. [12]. Then the experimenters can easily access this reconstructed data and perform their analyses. A sample analysis can be conducted with the obtained data by measuring the Z^0 mass which is calculated by the following equation:

$$|p_{Z^0}| = |(p_{\mu,1} + p_{\mu,2})| = m_{Z^0} \tag{1.1}$$

To get a statistically sufficient amount for a scientific statement of the Z^0 mass, this calculation has to be done for a huge number of like reactions. It

[1] Plus all known parameters of the accelerator, e.g. the center mass energy $\sqrt{s}$, and all parameters of the detector, e.g. which trigger has fired.

also needs to be considered that the desired reaction has to be singled out of all other collected data, which contains of course many other reactions. To perform this simple example of a physics analysis, it is sufficient to write a computer program for it.

1.2 Programming with C++

Today's standard programming language for physics analyses is C++. This has some major reasons:

- C++ programs can be optimized to run very fast.
- Programming with C++ supports the two programming styles: procedural and object-oriented.
- It is platform independent. This means, everybody can create a software package or a class library and distribute the source code. A favorite class library for example is ROOT[2].

But performing an analysis with a self-written computer program has also its disadvantages. First of all, the user has to learn a complex programming language which is normally not the field of application for a physicist. Another effect is that the physicist might loose the focus because changes or new implementations of the analysis need more time to translate them into code. In addition, the produced source code has to be maintained and can not easily be exchanged with other experimenters.

1.3 Graphically designed analyses

The opposite of a pure source code based analysis is a graphical designed analysis. An example for such an environment is Lab View[6] (mainly used in electrical engineering), where no learning of a complex programing language like C++ is required. Here, a developed analysis gains more clarity through the graphical visualization and improves the exchangeability through a own data format of the analysis a lot.

1.4 Applications in a web browser

A further development of graphical applications is driven by the advance of the Internet. It's so-called Web 2.0 technologies[24] opens new possibilities

for various web applications (apps), which run in every standard web browser. For example in figure 1.2, a web mail client, DIMP[16], is presented.

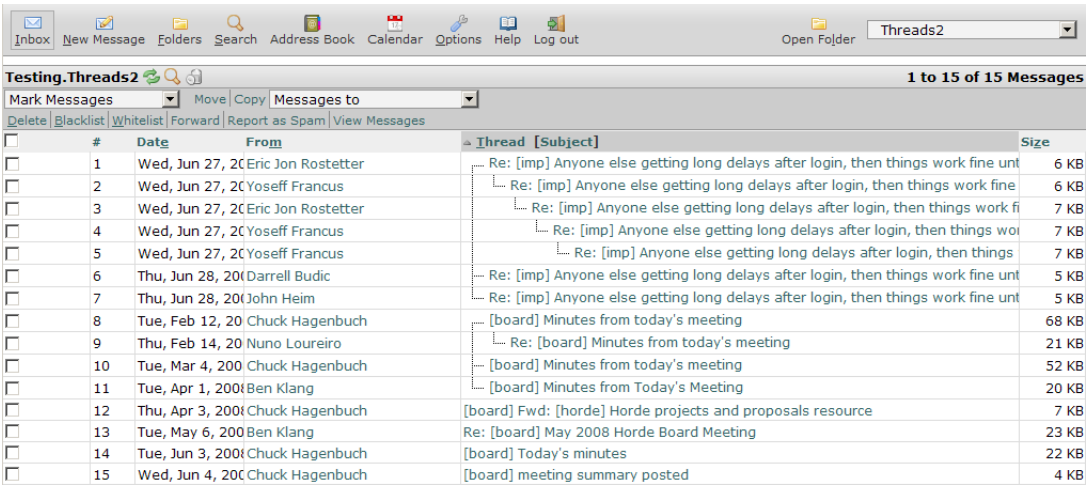

Figure 1.2: Web mail client: DIMP

The following advantages of a browser based application in contrast to a graphical application/ self-programmed analysis make this concept very attractive for physics analyses.

- There is no need for an operating system based installation on the client. Only prerequisite is a web browser supporting JavaScript[2].
- The user needs no administrator rights to run a new application.
- An application runs on the client with only the front-end. The back-end runs on the server. Therefore, the application uses the server computing resources and becomes independent of the client's computer hardware/environment.

In this work, I will demonstrate, how these new technologies can be used for developing a graphical physics analysis environment as a web application and how to perform a physics analysis with it.

[2]The exact support for a feature is specified in the supporting HTML standard of a browser.

2 The Technology and Environment

2.1 The PXL class collection

The Physics eXtension Library (PXL)[30] will be used as a core component for easy handling of the analysis itself. PXL is a class collection which is written in C++ with only a few dependencies like ZLIB[29] and EXPAT[3]. It supports the creation of physics analysis for HEP and astroparticle physics with all necessary data objects. Between these objects it is also possible to create different types of relations, e.g. mother-daughter relations between particles to store decay trees. I used PXL 3.0 which has the following main features and is divided into these packages:

- 'core': This package includes classes for input, output and a basic event class (`pxl::Event`) which provides access and handling of the data. The core contains also a class for four-vector analysis (`pxl::LorentzVector`). A specialty of PXL is that the user can attach additional analysis specific information (so-called user records) to PXL classes like the `pxl::Event`.
- 'modules': The `pxl::Analysis` class is provided for modeling and running an analysis which consist of modules that are connected to each other. This package includes some standard modules like the `pxl::InputModule`, `pxl::OutputModule` and the `pxl::PyAnalyseModule`. These modules are supplying the logic of the physics analysis and thus improve its clarity. The actual analysis code is packed into the modules by a Python script. The user is also entirely free to use self-written C++ modules.
- 'hep': For HEP there is a particle class (`pxl::Particle`). A particle object is able to contain all standard information like energy, momentum, pseudo rapidity and charge. In addition, the user can also attach more information through user records. A main feature is that in this package it is possible to define different views (`pxl::EventView`) of the same event. In a Monte-Carlo simulation this can be a 'generator' view, e.g. what event was generated, and a 'reconstructed' view, e.g. the event after it

passed the detector simulation.

- 'astro': PXL provides support for astroparticle physics analyses. Amongst others, there is a cosmic ray class (`pxl::UHECR`) which can be connected to a source (`pxl::UHECRSource`). Furthermore, the `pxl::CoordinateTransformations` class can be used to transformation between astronomical coordinate systems.
- 'algorithms': This package contains some algorithms for HEP and astroparticle analysis. For example, there is a algorithm for automatic decay tree reconstructing by presetting a steering file (`pxl::AutoProcess`). Another very useful algorithm is the `pxl::AutoLayouter`. The common use is the layouting of decay trees in a HEP event with this algorithm. For more information and how the algorithm works can be found in Ref. [27].

The procedure for building a physics analysis with PXL is defined by the modules in the analysis class. Each module can have different sinks (input) and sources (output) which when being connected with other module´s sinks and sources, create an analysis flow. During a running analysis, PXL events are passed along these connections to the different modules for analysis.

A module's logic can be controlled by specifying its configuration options. This can be for example the path of an input file or a Python script. The use of the programming language Python has the advantage that it is much easier to learn than C++ and therefore supports fast prototyping.

If the user is not interested in creating his own C++ modules for PXL and does not want to install a C++ compiler, there is a Python wrapped version of PXL called PyPXL. It is generated with the help of the SWIG software tool. For more information about SWIG see Ref. [23]. The user can write his analysis in Python code but nevertheless benefits from the underlying performance of C++ in intensive computing program sections like IO. This concept becomes even more attractive when also using PyROOT besides PyPXL for plotting and histogramming[2].

Another feature of PXL is its ability to save the `pxl::Analysis` in a single XML file. In general, the information that gets saved is only the used modules, their option's values and the connections among them. For additional information, it is possible to add user records to each module and to the analysis class itself, too. Such information can be the author's name of the analysis or the compilation date of a module. The user can also load a `pxl::Analysis` from a created XML file. To execute an analysis as a batch job, PXL delivers the `pxlrun` program which only needs such a XML file to run the formerly created

analysis.

In conclusion, the PXL class collection is well suited as a core component for physics analysis because of its fast C++ code, its easy usage in Python and the clear representation of the analysis logic by the modules.

2.2 The VISPA development environment

The Visual Physics Analysis (VISPA) program is a graphical development environment for designing physics analysis. It is programmed in Python and uses PXL as an underlying analysis framework[10, 11].

Figure 2.1: VISPA: Logo

In the main window of VISPA, the abstract PXL modules become now visible. A sample analysis is shown in figure 2.2.

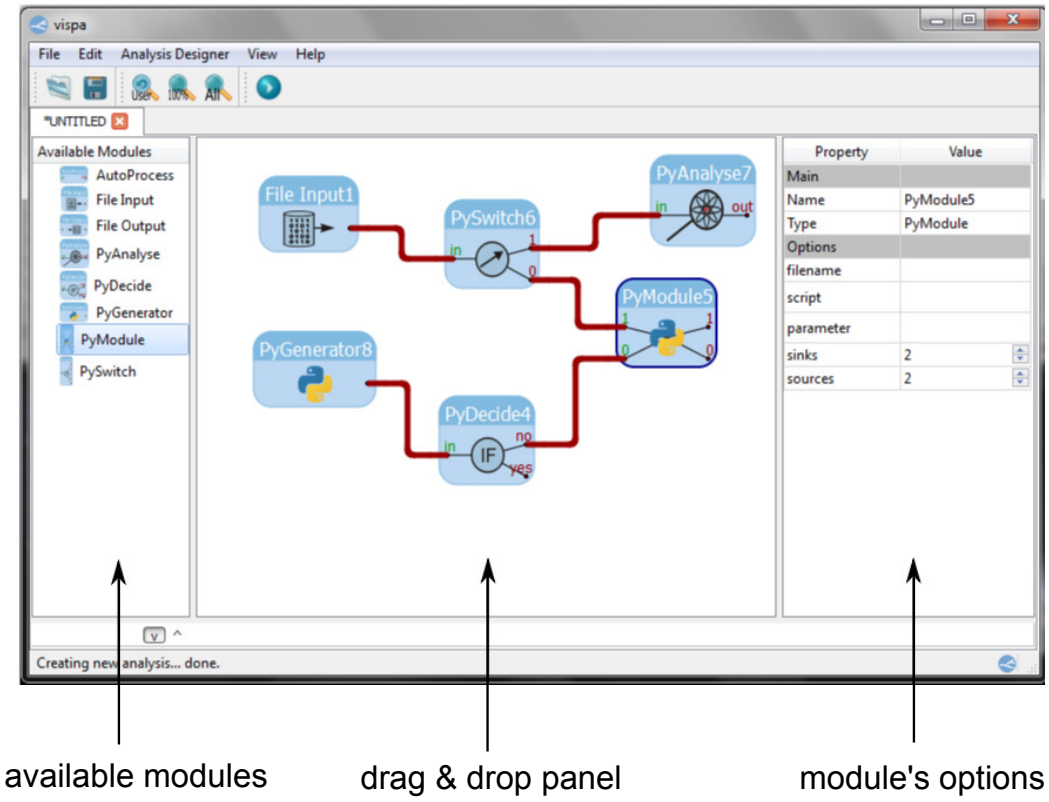

Figure 2.2: VISPA: Creating an analysis

On the left, there are all available C++ and Python modules listed. Per drag and drop, the modules can be moved into the main panel. Here, the dropped module will be created and drawn with its sinks and sources. By selecting a module, its configuration options are displayed and can be edited in the right panel. Connections can simply be made by dragging a source over a sink or vice versa. To edit a Python analysis script of a module, a simple double click opens your standard data editor for Python scripts. If you double click on an input or output module instead, it opens the data browser, a further feature of VISPA.

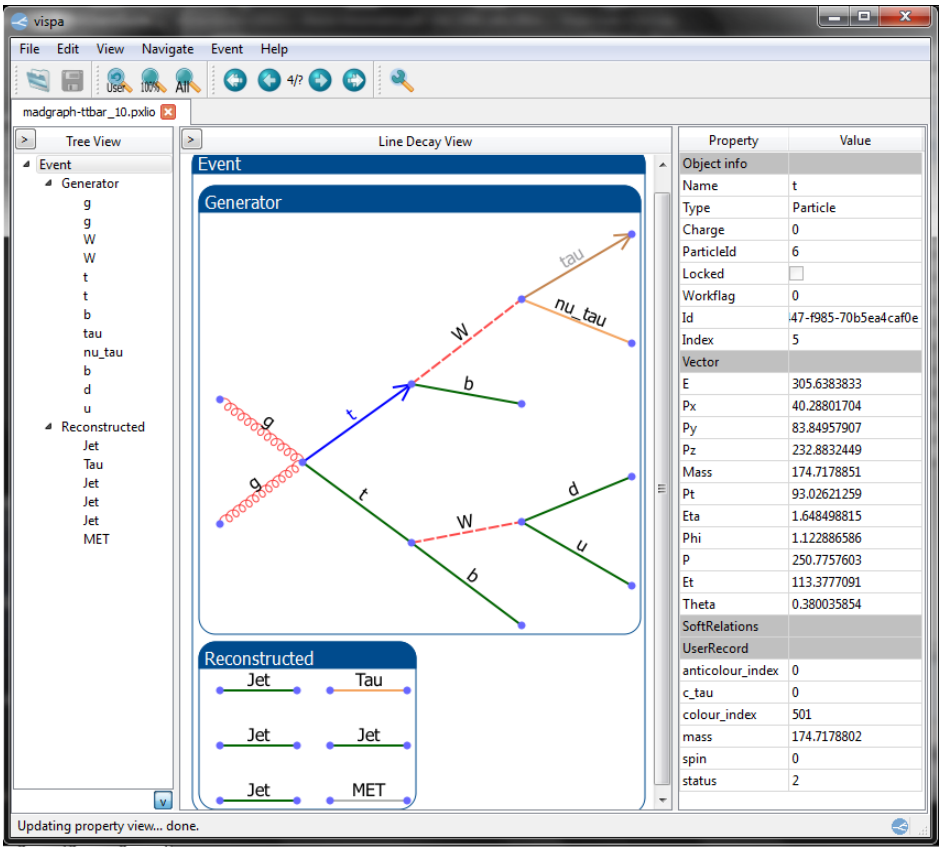

Figure 2.3: VISPA: Event viewing with the data browser

Figure 2.3 shows the data browser with an event which is an example of a Monte-Carlo data set with two different event views, 'Generator' and 'Reconstructed', of the same event. If the user selects a particle, its properties are displayed in the property panel on the right. Furthermore, the `pxl::AutoLayouter` algorithm has been used to draw the decay tree in the center panel.

Another feature of VISPA is its ability to save a designed analysis in a XML file. This makes a physics analysis easily exchangeable and distributable

among teams of physicists, who are working on the same analysis or have interdependent analyses.

To sum it up, VISPA shows a desired graphical implementation of a physics analysis development environment with all its advantages that come with the underlying PXL framework and extends it with GUI components like the event viewing with the data browser.

2.3 Combining with Web 2.0 technologies

An implementation like the VISPA environment to perform a PXL analysis is not limited to a stand-alone application. The new technologies that are associated with the Web 2.0 have been taken under consideration and proven to be useful as a next step in development of physics analyses. The task is to create a development environment for PXL physics analyses, which consists of a front-end in a web browser but its underlying analysis is stored and executed on a server. I choose to program the web server in Python for easy access to the analysis with PyPXL and JavaScript (JS) on the client because it is the most supported language in the major web browsers. Alternative technologies besides JavaScript, might be Java applet[26], Flash[19] or Silverlight[5].

For the GUI, the following JS libraries were used:

- 'Draw 2d': This library provides a special canvas, a workflow, on which HTML elements can be positioned, moved and resized. In addition, it contains necessary classes which can be used for drawing UML diagrams or mind maps[17]. It depends on the JavaScript 'VectorGraphics' library.
- JavaScript 'VectorGraphics': This package wraps around the canvas object, which is specified in HTML 5.0[18]. It is used for drawing vector graphics like lines, polygons and circles with different stroke styles and colors[31].
- 'Ext JS': It is an entirely JS based framework, which provides GUI components like text fields, buttons, menus, trees and grids in an object oriented structure. Thus, it is highly flexible and extensible. Furthermore, there is support for managing data objects like data stores or component layouters. I have used the version Ext JS 3.2.[20]
- 'MooTools': This is a framework which defines new convenient functions to JS. For example, the '`$`' operator is introduced to access HTML components via the DOM structure of a website by its id. [28]

For communication between client and web server, I used the JavaScript Object

Notation (JSON)[7] format because it is a human readable and easy writable data exchange format. JSON itself is capable of storing primitive data types like strings, numbers and booleans into the two data structures: arrays and objects. JSON objects are defined by key and value pairs. Other alternatively formats are XML or Jamal[8].

Communication will be asynchronously by a POST request which is send to the sever via HTTP. Then the server sends a response to the request back to the client. This concept of communication is commonly associated as a part of "Asynchronous JavaScript and XML (AJAX)". Figure 2.4 clarifies this in comparison to traditional synchronously applications.

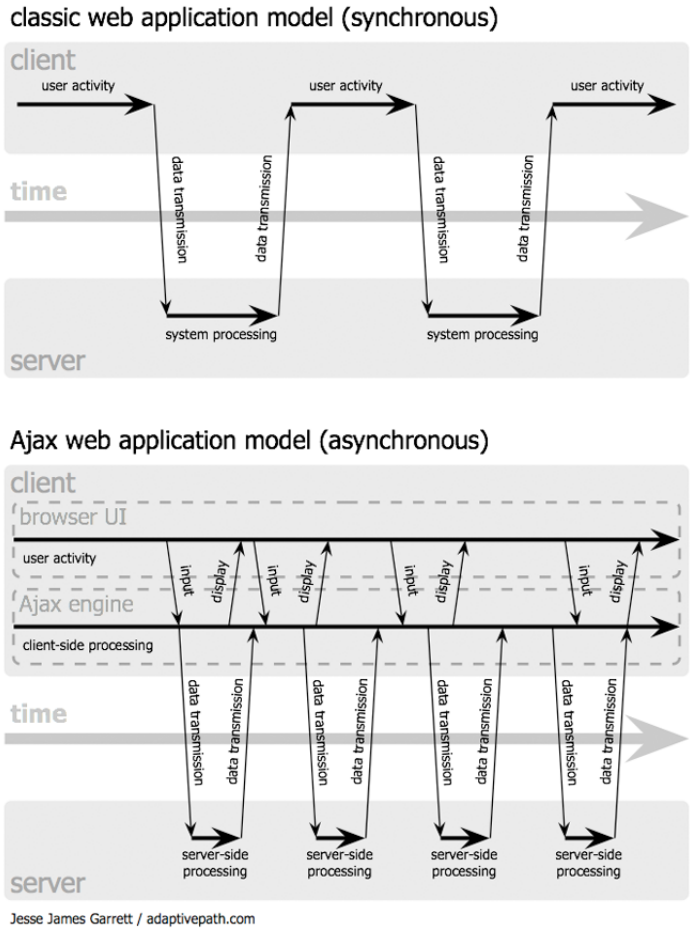

Figure 2.4: The difference between traditional and AJAX applications
The figure was taken from Ref. [21].

A JS script on the client can send a request via the AJAX engine to the server. The server will now progress the request while the client has not to wait for a response. If the client has specified a function within the request,

which to be call after a successful response, the AJAX engine will take care of it.

These are the technologies which will serve the creation of a development environment for physics analysis as a web application. The project is called 'VISPA@WEB'.

3 The Implementation of the Server

The main part of the program is the web server. Its implementation has to receive and process any incoming GET or POST request from multiple clients simultaneously. First, the server has to identify active clients and match them with their physics analysis. This is solved by a server side generated id. Once a client sends a creation request for a new analysis to the server, it will receive and store this id. All subsequent requests have to send this id along with the request data to modify the correct analysis.

In this context, I will call a physics analysis with its id also an analysis session. The user can have more analysis sessions at a time which are distinguished by different ids.

An example of a request is the following:

```
{
   request:   ‘‘saveAnalysis’’,
   filepath:  ‘‘user/ttbar.xml’’,
   session:   ‘‘8ad2bf6754dfe153...’’
}
```

The analysis session id is actually a SHA256 hash of the timestamp at the generation time plus a random number. The id generation is easily exchangeable which makes this feature adaptive.

Figure 3.1 visualizes the data flow and processing of a request on the server and client side.

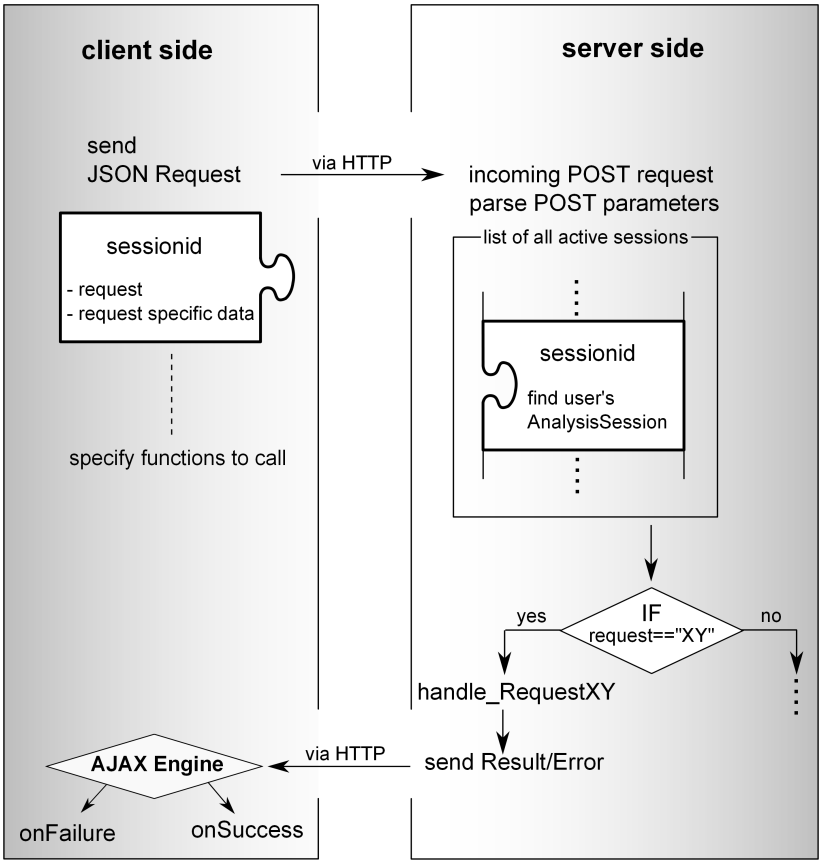

Figure 3.1: Request management between client and web server

At first, the incoming request will be examined whether the content-type is specified as

`application/x-www-form-urlencoded`,

which is the common signature of a JSON request. In this case the server extracts the JSON object's key and value pairs. To find the client's analysis session, the id that was sent will be compared. The web server delegates the request to the corresponding method for further processing. This method will take care of all necessary actions and send the results back to the client. In the example above, a method is called to save the `pxl::Analysis` to the specified file path, `'user/ttbar.xml'`.

If an exception occurs during the processing of a request, the server can send an error message back to the client. This message is also passed to the `onSuccess` method of the client by the AJAX engine. Therefore, it has to be distinguished from successfully returned results.

The AJAX engine will call the `onFailure` method, if the POST request has

not reached the server or no response has been send in the timeout. This may happen for instance, if the connection between them is abruptly lost. So, in fact, the client has to manage two different levels of exception handling.

As stated above, every client needs a valid id to get a request processed. Therefore, I like to elaborate what exactly happens when a new client connect for the first time to the server and how it receives an id with an associated analysis session. Here for security reasons, a major requirement was to realize different environments for a guest and a registered user. The flow chart in figure 3.2 demonstrates the chosen implementation.

At first, a login with a username password pair is required to distinguish between the two user types (1). A guest needs no password and receives after the successfully returned login request his analysis session id (2). If a user is registered and enters his username and password, both will be sent to the server for authentication (3). If the user has logged in before the response of the server includes all of his previously saved XML analysis files. He can create a new physics analysis (4) or load and work further on an existing one (5). In both cases, the server will send an analysis session id back.

For security reasons, the password is not transmitted in plain text. At the start, the client will save the current timestamp. The entered password will be concatenated to the timestamp and a SHA256 hash is calculated. The server will receive both, hash and timestamp, and can then easily generate and compare the stored user's password by computing the same hash.

3.1 Analysis session management

For working with many different analysis sessions simultaneously, a system is needed that controls and organizes analysis sessions automatically. The concept for handling a single analysis has already existed during the work with PXL 2.5.6 and therefore the `pxl::Analysis` class was introduced with PXL 3.0. For more flexibility, I created the `AnalysisSession` class to wrap methods of the `pxl::Analysis` class and to implement more concepts like the analysis session id or the file handling with the file system.

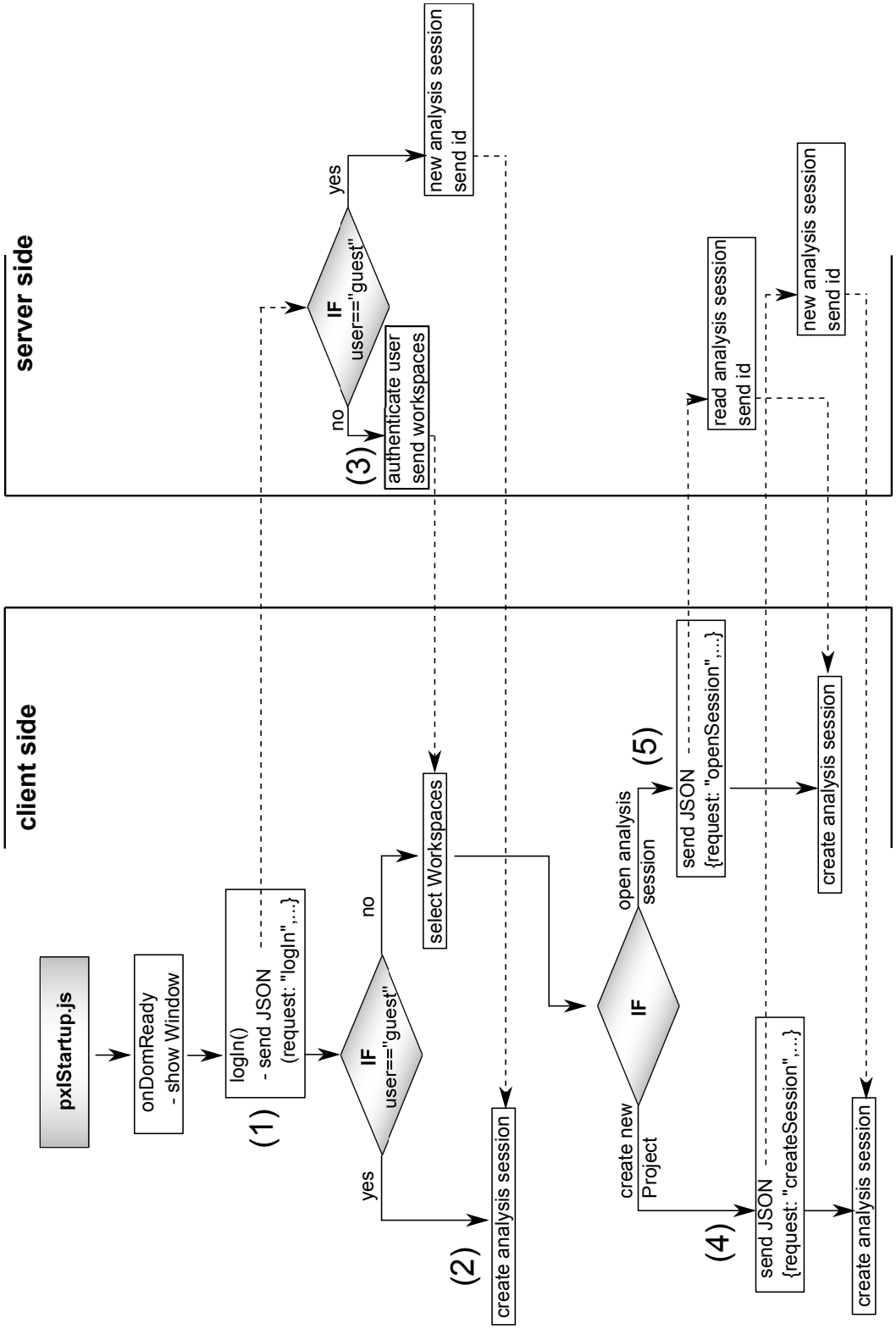

Figure 3.2: Client: Getting an id

It is convenient to introduce a pool where all active analysis sessions are maintained. In this context I distinguish between an active and a dead analysis session. All analysis sessions have a timestamp of their last modification time. To save memory and prevent the loss of data if the program crashes, analysis sessions can be saved to disk and removed from the analysis session pool if they are not modified for a specified time. These analysis sessions are then marked as dead.

Figure 3.3 demonstrates the interaction of the pool and the analysis session class. For clarity, methods which support modifications, like the creation of new modules for the `pxl::Analysis` object, are left out in this diagram.

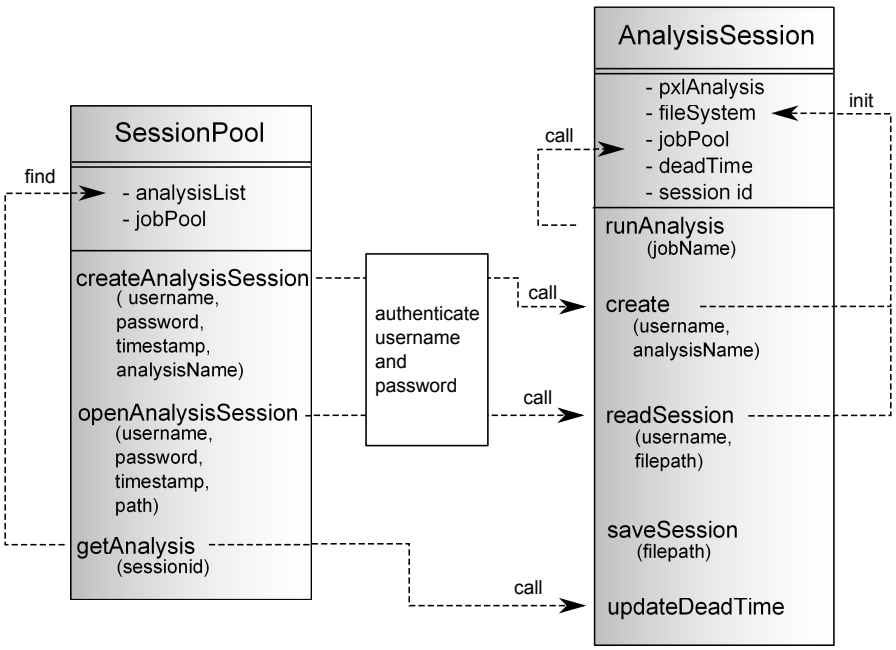

Figure 3.3: Interaction between `SessionPool` and `AnalysisSession`

The analysis session pool manages mainly the creation of new analysis sessions. It can create an empty analysis session or generate an analysis session from a XML file. If an analysis session is requested by the `getAnalysis` method, its timestamp will be updated and moved to the bottom of the `analysisList`. This has the effect that this list is automatically sorted by the time of inactivity. So, possible dead analysis sessions can easily be found at the top.

3.2 File handling

PXL provides its own data format with the extension, 'pxlio'. The structure of a pxlio file is divided into chunks. Figure 3.4 shows an example of a possible constitution for a pxlio file with two chunks.

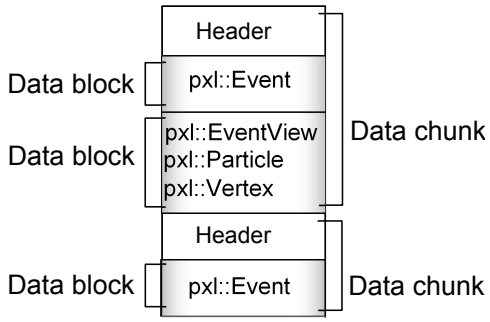

Figure 3.4: Possible internal structure of a 'pxlio' file

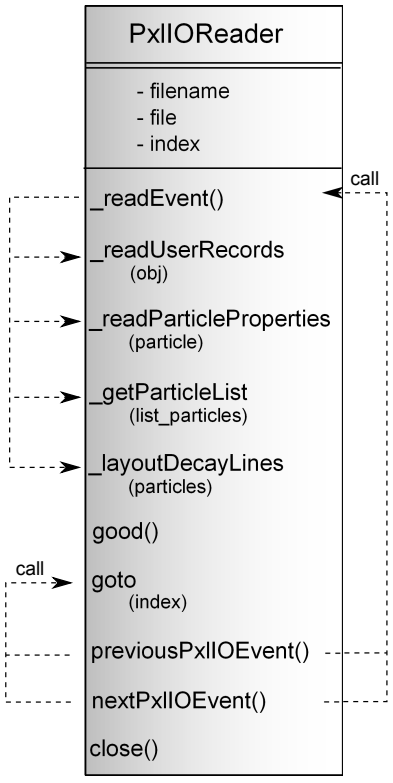

Figure 3.5: The `PxlIOReader` class

In a pxlio file, there exists no global header which leaves the file readable

even when the writing process is interrupted. Each chunk consists of data blocks with one or more serialized objects.

To access this information and to send it back to the client in an easily parsable format, I implemented the `PxlIOReader` class. Figure 3.5 shows the current class structure.

The `_readEvent` method extracts the information of the current `pxl::Event` at the `index` position of the file and writes it into in the format which is shown in figure 3.6.

```
{
  EventID
  records <array>
        {name, value}
  EventViews <array>
        {
          Name
          Particles <array>
                {
                  Name
                  id
                  prop <array>
                     {name, value}
                  records <array>
                     {name, value}
                  x1, x2, y1, y2
                }
          records <array>
                {name, value}
        }
}
```

Figure 3.6: The data format of the `PxlIOReader` class

This format is highly flexible because particle properties are saved in arrays of property names and values. Therefore it can easily be adapted to more or less information that needs to be send. For example, if a `pxl::Event` includes over hundred particles the user might not be interested in all information available.

If all arrays of name and value pairs are empty, the format contains the minimum of possible information but the client is still able to process it.

3.3 Job management

For performing parallel analyses, a multiprocessing job management is needed to prevent the web server from crashing if an analysis breaks down. I created a job pool similar to the analysis session pool. Every user can submit an

analysis to this pool for execution. Figure 3.7 displays the complex submission and execution process.

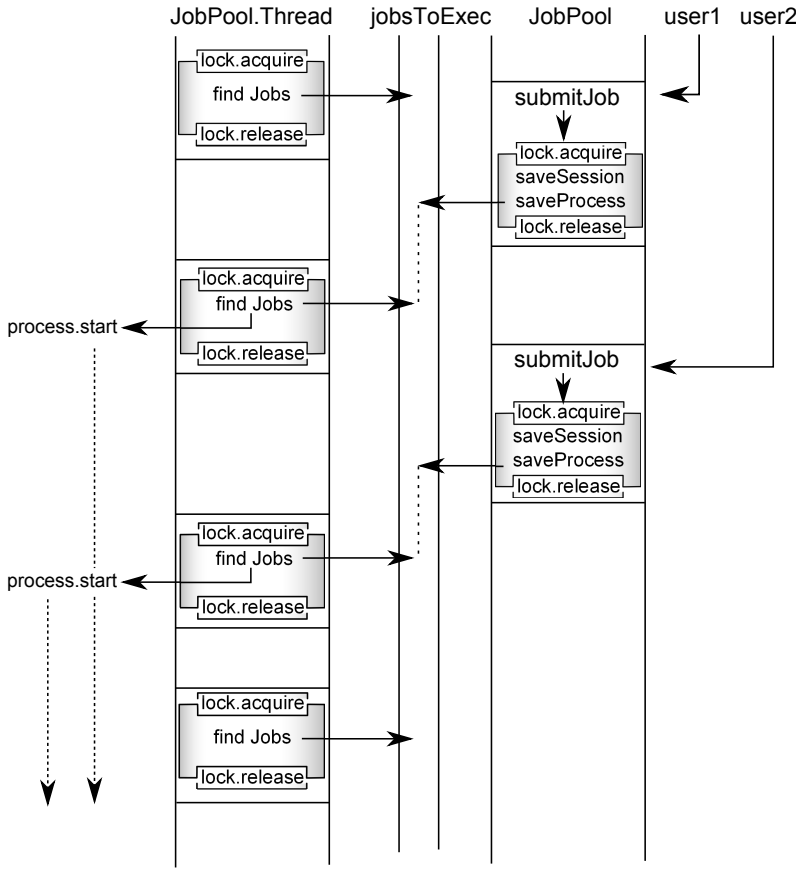

Figure 3.7: Multiprocessing with the `JobPool` class

The pool runs mainly in a separated thread to control the maximum number of parallel working processes. Submitted jobs will be enqueued and wait for their execution. Every second the pool will look for new jobs in that list and execute them if the maximum number of parallel jobs is not yet reached. Otherwise the jobs will remain in that list until an already running job is finished. It is convenient to set the number of the allowed parallel running jobs to the server's core count for an effective use of the available computing resources.

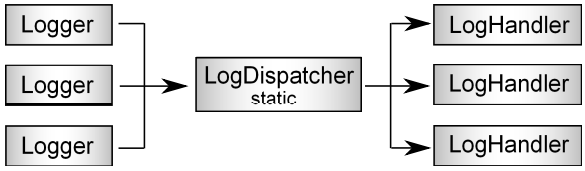

Figure 3.8: The logging system of PXL

The output stream of a job can be easily caught by using the new logging mechanism of PXL 3.0. Figure 3.8 demonstrates this new feature and how it works.

To send a log entry, a `pxl::Logger` object has to be initialized with the module's name. The `pxl::Logger` can be called with a specific log level and the message to log. The global `pxl::LogDispatcher` will only distribute incoming messages from a `pxl::Logger` to all registered `pxl::LogHandler` up to the highest set log level by the handlers.

I created the `HTMLLogHandler` class for catching the output and displaying it to the user. The class inherits from the `pxl::LogHandler` and is registered to the `pxl::LogDispatcher`. It generates a HTML page from the incoming log messages and displays it color-coded by the log level. To distinguish between the logging of different users, the `pxl::LogDispatcher` exists for each job process separately. This was also an argument for the implementation of a multiprocessing instead of a multithreading job management.

4 The Graphical User Interface

4.1 Module Handling

The modules are managed by using the 'Draw 2d' library. Figure 4.1 shows all modules and what they look like.

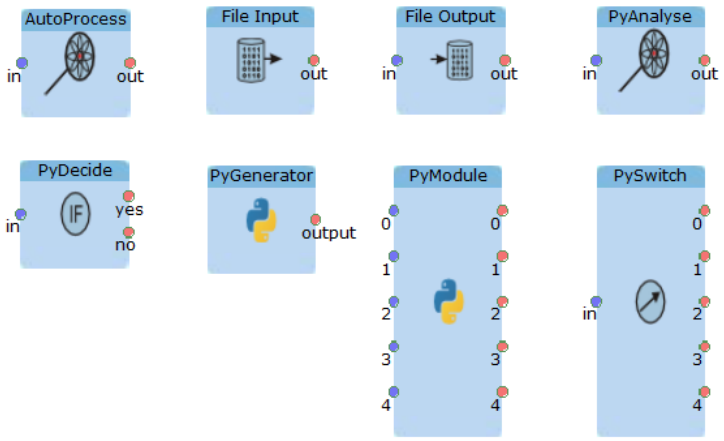

Figure 4.1: Overview of all default available PXL modules

A module can be created by drag and drop, selected, moved or erased on the provided workflow of the library. sinks and sources are added to them based on the module's description. The sink and source classes inherit from the `draw2d.Port` to react on the user, if they are dropped on each other. Then a connection is established by sending a request to the server with the two module's and port's names.

An example of a created analysis is shown in figure 4.2.

The main goal of the client's implementation was to adopt the VISPA layout and its handling into the web browser. Here, the 'Ext JS' library provides the necessary graphical components to satisfy even more complex requirements like a tab panel under the menu or the module's options editor with a property grid on the right.

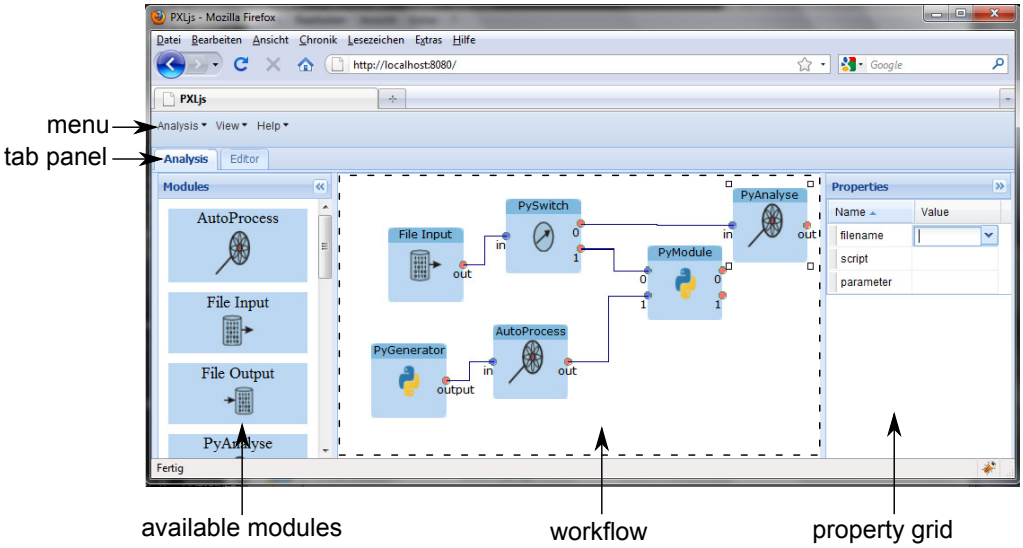

Figure 4.2: A sample analysis

4.2 File browsing

For security reason, the user should be restricted to perform his analysis only in his own directory. Also, the user should not see the real file system structure. Therefore, I created a virtual file system manager between the client and server which will manage the user's files and displays only specified file extensions. The underlying security concept was not to implement complex rules of what the user is not allowed to do. It is intended to only provide what the user is supposed to do or see.

Figure 4.3 demonstrates an example how the file system works.

Consider a user who likes to set a module's filepath to a Python script which is located on the server. The client will send a request to the server to set the given file path (1), e.g. the virtual path `'user/test.py'`. This file path will be translated to the real file system path on the server, e.g. to `'C:/work/test.py'` (2), and will then be applied to a specified module (3). The client displays the virtual set path correctly because the real path is retranslated by the virtual file system (4), in this case to `'user/test.py'`. Therefore, the client has no knowledge of the real file system. Another advantage of this concept is that it can be simply adjusted to different operation systems or file system environments.

To display this virtual file system to the user, I programmed a file browser which is shown in figure 4.4. The file browser enables the user to organize his files on the server. It is even possible to download or upload a Python script.

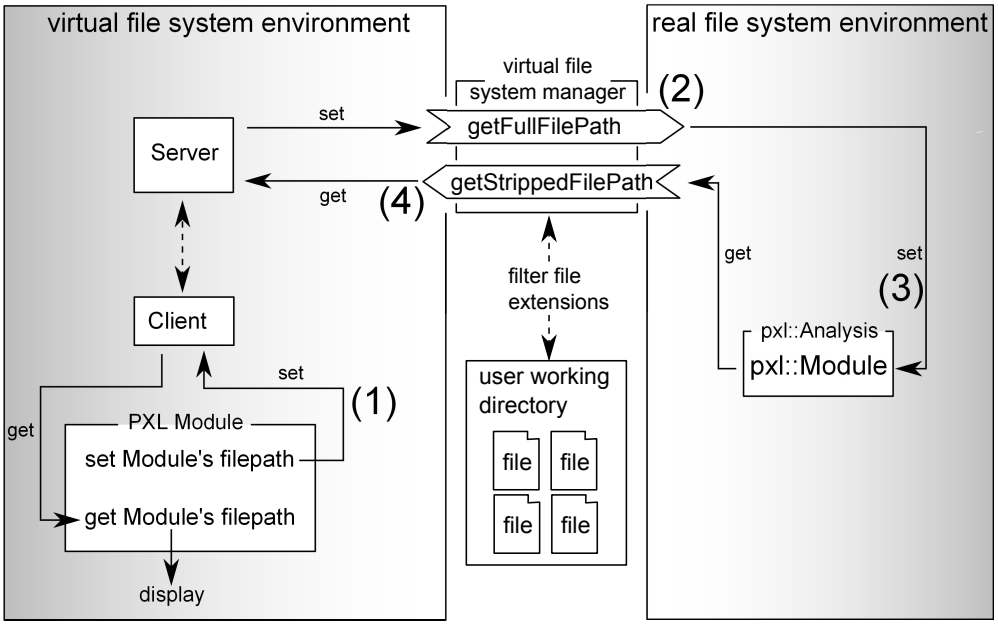

Figure 4.3: The virtual file system

The maximum size of a file to be uploaded can be restricted. If a group of users likes to work with the same data set, the file system differs between a user specific folder (`'user/'`) and an additional globally read-only folder (`'global/'`) which is visible to all other users. For editing, the user can copy files with the file importer from the global folder to his own one. The implementation is also flexible what concerns adding files from other designated import directories. The file importer is shown in figure 4.5.

Files can be chosen from the left via the 'add' button and are then automatically get copied in sequence by pushing the 'start import' button.

4.3 The Data Browser

In a similar manner like VISPA, I also used the `pxl::AutoLayouter` to display decay trees in a data browser as in figure 4.6.

On the left side is an overview of all event views and all contained particles of the current event. Above it, the index of the current event in the file is displayed. It is possible to navigate forward or backward to display the next or previous event. The right panel displays the properties and user records of the selected particle, event view or of the whole event. The shown information is actually the array of name and value pairs of the `PxlIOReader` class interchange

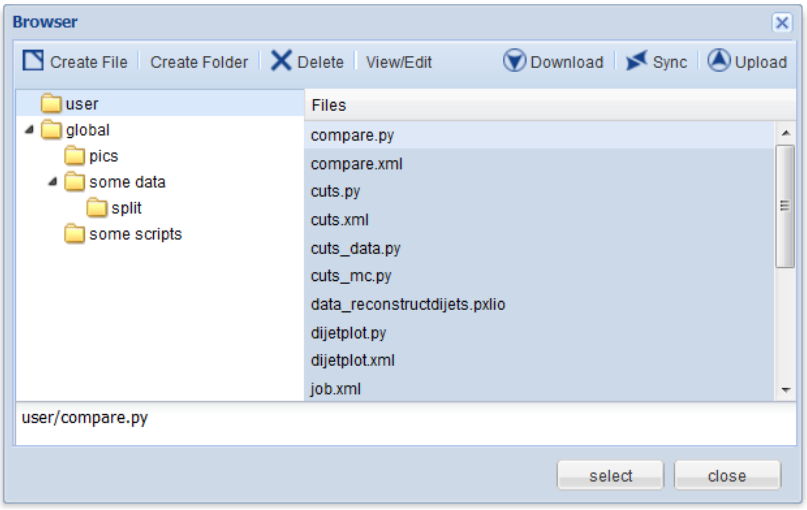

Figure 4.4: The file browser

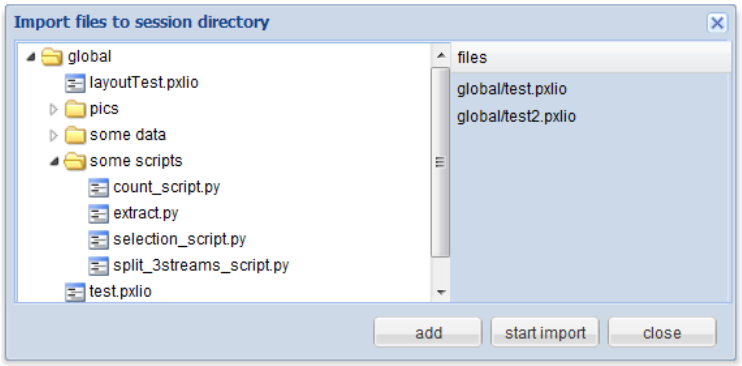

Figure 4.5: Implementation of the file importer

format.

To draw the decay lines, the JavaScript 'VectorGraphics' library was used. I also implemented a lightweight server side `ParticleLine` class to use the `pxl::AutoLayouter`. If a particle has no mother and daughter relations, the line of the particle is adjusted horizontally and in sequence to other particle lines as in the 'Reconstructed' event view in figure 4.6.

After the `pxl::AutoLayouter` has been executed, the positions of the decay lines are included in the `PxlIOReader` format as well and transmitted along with the other data of the event to the client.

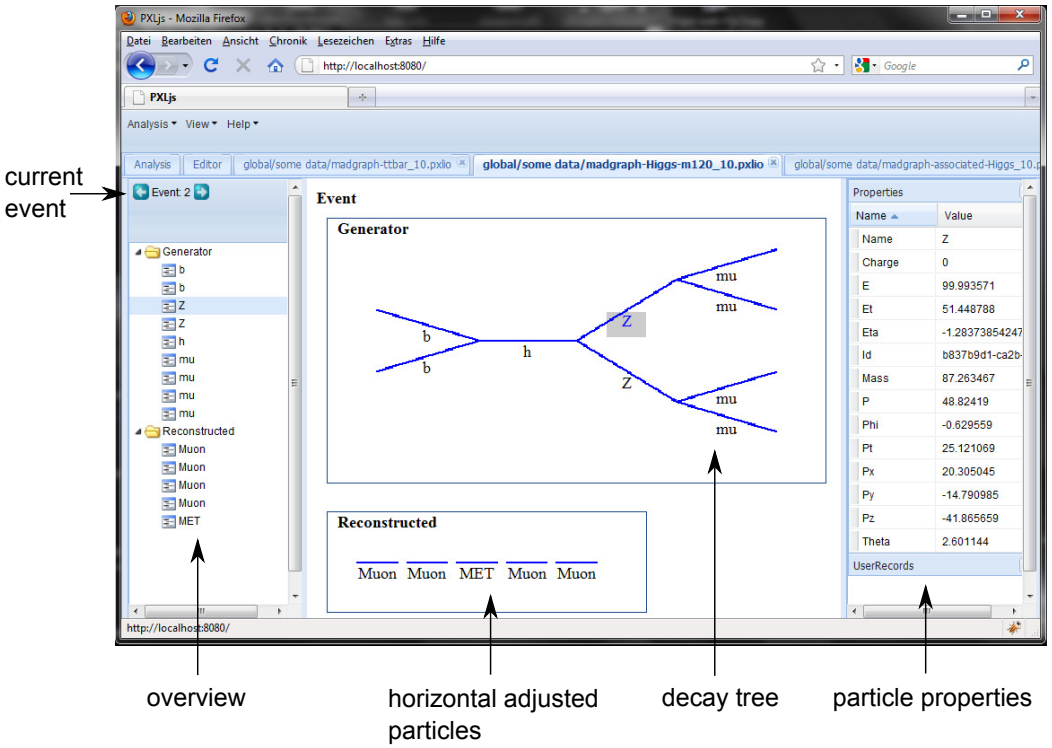

Figure 4.6: Client's data browser for event viewing

4.4 The Code Editor

Unlike VISPA, the user can edit Python scripts directly inside the web browser. The code editor, EditArea[9], was therefore integrated into the GUI. It supports syntax highlighting and interacts with the file browser to open or save a file. It is also possible to undo a modification or change the font size of the displayed code. It is left to the user, if he likes to use this feature or if he likes to downloads the Python scripts and edits them with his own editor. Figure 4.7 visualizes the integration of the editor into the 'Ext JS' layout.

To summarize, the main layout of VISPA and its handling could well be transferred into a web browser. It was even possible to adopt complex features like the data browser. In addition, for working with the server's file system, a file browser and file importer was designed and implemented.

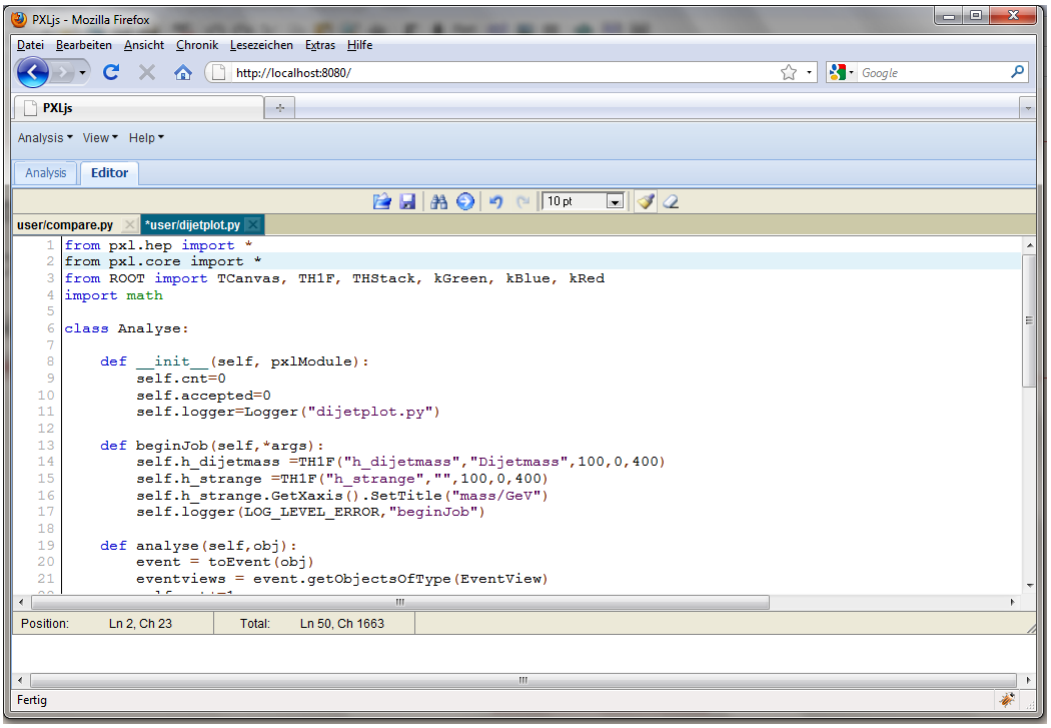

Figure 4.7: The code editor: EditArea

5 A Demonstration: Dijet Mass Measurement with CMS

To demonstrate the potential of the developed web browser based analysis environment, I measured the dijet mass of the recent data taken by the CMS detector at the LHC.

5.1 The LHC

The Large Hadron Collider (LHC) is a proton-proton collider ring at the European Organization for Nuclear Research (CERN) in Geneva, Switzerland. It is constructed in the former LEP tunnel and has a circumference of approximately 27 km. It is designed for a center-of-mass energy of $\sqrt{s} = 14$ TeV. One proton beam consists of up to 2808 bunches, each with $1.1 \cdot 10^{11}$ protons (at injection) and collides with an average bunch-crossing rate of 31.6 MHz. This yields to a luminosity of $L = 10^{34} \frac{1}{cm^2 s}$ and roughly 600 million collisions per second. It is also possible to perform heavy-ion experiments with an energy of 2.76 TeV per nucleon. The following experiments are conducted:

- ALICE: A Large Ion Collider Experiment is designed especially for heavy-ion collisions like Pb-Pb. It is used to study the properties of quark-gluon plasmas.
- ATLAS: A Toroidal LHC ApparatuS, it has a special donut-shaped magnet system and is used in a wide field of analyses, from Higgs physics to supersymmetry (SUSY) and extra dimensions.
- CMS: Compact Muon Solenoid, it has the same goals like ATLAS but it was built with other technical solutions and design.
- LHC-b: Large Hadron Collider beauty Experiment, this examines the asymmetry between matter and antimatter with a forward sub-detectors system.

In figure 5.1, the CERN accelerator complex with the LHC and its experiments is shown.

To keep the proton beams on track, approximately 10,000 magnets were installed. The dipoles for bending the beam are supplied with a current of nearly 12 kA and create a dipole field of 8.33 T at the peak. These dipoles consist of superconductors and are cooled down to 1.9 K with superfluid helium [15, 4].

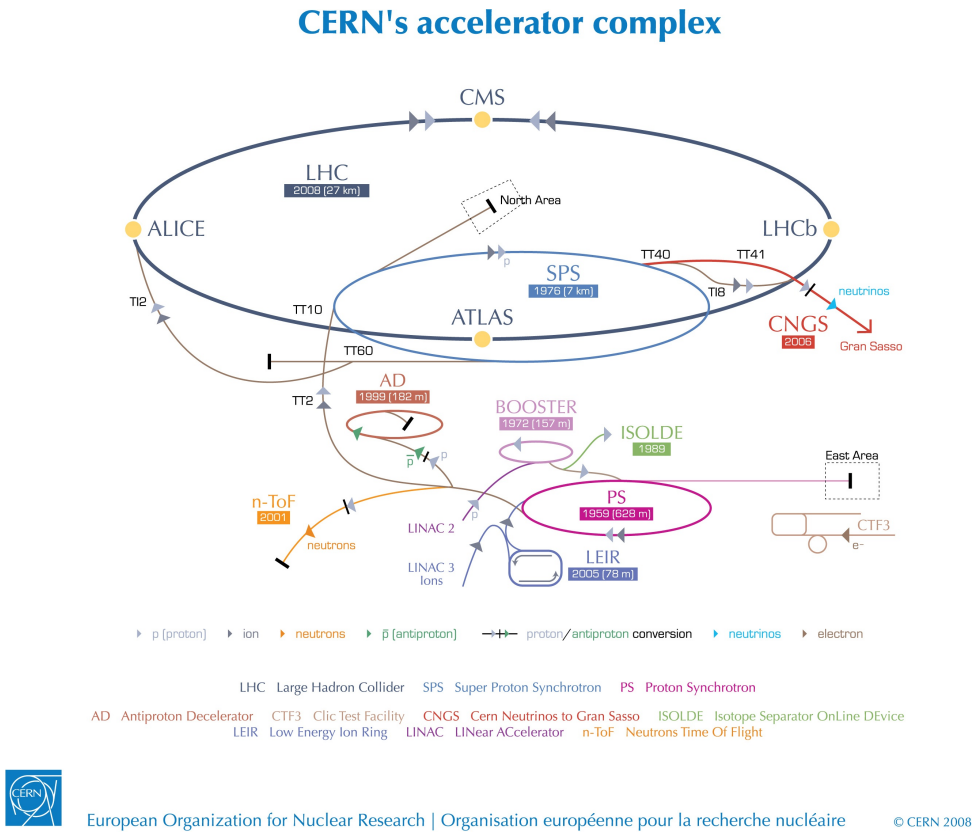

Figure 5.1: The CERN accelerator complex
Figure was taken from Ref. [25].

5.2 The CMS detector

The Compact Muon Solenoid, is a cylindrically shaped general purpose particle detector at the LHC. Its composition is shown in figure 5.2. A major difference to the ATLAS detector is the magnetic system to measure charged particle's and especially muon's momenta. For high resolution, the solenoid superconductor provides a magnetic field of $B = 4$ T . The layout of the detector covers measurement of particles with a pseudorapidity of $|\eta| < 3$ with the electromagnetic calorimeter and $|\eta| < 5$ with the hadron calorimeter[4].

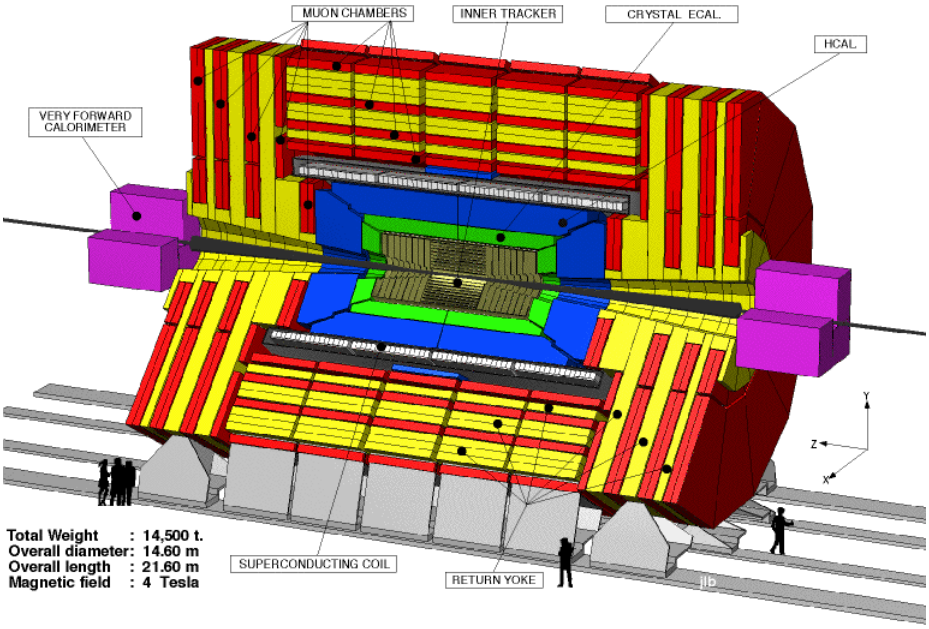

Figure 5.2: The CMS detector
Figure was taken from Ref. [22]

The main parts are:

- Muon system: Muons are measured at three locations: First in the inner tracker, then after the coil and in the return flux of the magnetic field. To satisfy a large detection surface in different radiation environments, the design includes three gas detector types:
  - In the barrel region with a low muon and neutron flux, drift chambers (DT) are used. It covers a pseudorapidity of $|\eta| < 1.2$.
  - Cathode strip chambers (CSC) are installed in the endcaps ($|\eta| < 2.4$), where the radiation level is high.
  - In both regions, resistive plate chambers (RPC) are used in addition for fast response and a good time resolution. However, their position resolution is less than the DT or CSC.
- Electromagnetic calorimeter (ECAL): It consists of lead tungstate (PbWO4) crystals in the barrel (61200) and in each endcap (7324). They are fast (80% within 25 ns) and radiation hard. Their readout requires photodetectors which have an intrinsic gain to compensate for the crystal's low light yield. In the barrel, silicon avalanche photodiodes (APDs) and in the endcaps, vacuum phototriodes (VPTs) are used to operate in the

magnetic field.

- Hadron calorimeter (HCAL): The HCAL surrounds the ECAL in the inner barrel. It was designed to suppress non-Gaussian tails in the energy resolution and improve the MET measurement by a good containment. It consists of plastic scintillator tiles with embedded wavelength-shifting (WLS) fibers that carry the light to the readout system of multi-channel hybrid photodiodes (HPDs).
- Inner tracking system: To keep track of the large amount of particles a the high luminosity even in heavy-ion collisions, a tracking system is installed which is divisible by the particle flux in distance r to the beam:

  - $r \approx 10\,cm$: Pixel detectors are located closest to the IP where the flux is highest with $\approx 10^7/s$. A pixel has the dimension $(100 \times 150)\,\mu m^2$.
  - $20\,cm < r < 55\,cm$: Silicon microstrip detectors with a cell size of $\geq 10\,cm \times 80\,\mu m$ are used.
  - $r > 55\,cm$: The outermost of the inner tracker consists of larger-pitch silicon microstrips with the dimension $\geq 25\,cm \times 180\,\mu m$.

Further information can be found in Ref. [4].

5.3 The Standard Model

The standard model of particle physics (SM) is build on the elementary particles shown in the tables 5.1 and 5.2.

Up till now, deep inelastic scattering experiments have not determined an inner structure or expansion of these particles and therefore are considered as point-like. In the SM, these particles are interacting with each other by coupling to three types of forces which are shown in table 5.3.

Table 5.1: The SM Leptons

Particle	mass/$\frac{MeV}{c^2}$	charge/e	weak isospin $I_3/\hbar$
Electron e^-	0.549	-1	$+\frac{1}{2}$
Electron neutrino ν_e	$< 2 \cdot 10^{-6}$	0	$-\frac{1}{2}$
Muon μ^-	105.658	-1	$+\frac{1}{2}$
Muon neutrino ν_μ	$< 2 \cdot 10^{-6}$	0	$-\frac{1}{2}$
Tau τ^-	1776.84	-1	$+\frac{1}{2}$
Tau neutrino ν_τ	$< 2 \cdot 10^{-6}$	0	$-\frac{1}{2}$

The masses are taken from Ref. [14]

Generations: $\begin{pmatrix} e^- \\ \nu_e \end{pmatrix}, \begin{pmatrix} \mu^- \\ \nu_\mu \end{pmatrix}, \begin{pmatrix} \tau^- \\ \nu_\tau \end{pmatrix}$

Table 5.2: The SM Quarks

Particle	mass/$\frac{MeV}{c^2}$	charge/e	weak isospin $I_3/\hbar$	Color
up u	$1.5 - 3.3$	$+\frac{2}{3}$	$+\frac{1}{2}$	r, g, b
down d	$3.5 - 6.0$	$-\frac{1}{3}$	$-\frac{1}{2}$	r, g, b
charm c	1.27	$+\frac{2}{3}$	$+\frac{1}{2}$	r, g, b
strange s	104	$-\frac{1}{3}$	$-\frac{1}{2}$	r, g, b
top t	$171.2 \cdot 10^3$	$+\frac{2}{3}$	$+\frac{1}{2}$	r, g, b
bottom b	$4.20 \cdot 10^3$	$-\frac{1}{3}$	$-\frac{1}{2}$	r, g, b

The masses are taken from Ref. [14]

Generations: $\begin{pmatrix} u \\ d \end{pmatrix}, \begin{pmatrix} c \\ s \end{pmatrix}, \begin{pmatrix} t \\ b \end{pmatrix}$

Table 5.3: The SM Forces

force	rel. strength	gauge boson	mass/$\frac{GeV}{c^2}$	range /m	coupling
strong	1	8 gluons g	0	10^{-15}	color c
electromagnetic	$\frac{1}{137}$	photon γ	$< 10^{-27}$	∞	hyper charge Y
weak	10^{-6}	$W^\pm$ Z^0	80.398 91.1876	10^{-13}	(weak) isospin I

The masses are taken from Ref. [14]

All interactions and particles are associated to a local invariant gauge theory

$U(1)_Y \times SU(2)_I \times SU(3)_c$. The gravitational force is not included in the SM because of its very weak relative strength of 10^{-40}.

The theory seems to describe results of elementary particle experiments correctly but there are still missing pieces. Some gaps in the SM are:

- The Higgs-mechanism predicts the Higgs boson which is not yet found.
- There may be an overall symmetry between the elementary particles and the gauge bosons.
- What is the nature of dark matter.

The LHC with its experiments might successfully answer some of these questions.

5.4 Measurement of dijet mass

Jets are created from reactions with final state quarks. A single quark can not be detected directly because it interacts strong on a small scale (quark confinement)[1] and therefore hardonizes quickly to mesons ($q\bar{q}$) and baryons (qqq). This hadron particle shower can then be measured. Jet algorithms are used to group an amount of detected hadrons together and reconstruct the original quark.

Dijets, which originate from two quarks, provide insight into QCD interaction. The measured dijet mass is a quantity for the center-of-mass energy of the interacting two quark system. It can be used for a better understanding and to verify the existing theory of this interaction.

In this analysis, I measure the dijet mass spectrum which I reconstructed from the two highest p_T jets in an event.

The data, I used,

`MinimumBias-Commissioning10-GOODCOLL-v8`

was collected in April 2010 at 7 TeV with the CMS detector. Triggered using a MinimumBias trigger to identify collision events, which included over 2,200,000 events corresponding to a luminosity of $\sim 1/\text{nb}$. This was compared to a Pythia QCD Monte-Carlo with full detector simulation of 200,000 events,

`QCDDiJet_Summer09-MC_31X_V3_7TeV-v1.`

The following cuts are applied on each event of both, data and MC:

- A leading and a sub-leading jet with $p_T > 30$ GeV
- On each jet:
  - $|\eta| < 2.6$

- EMF> 0.01 (electromagnetic energy fraction of the jet)
- n90Hits> 1 (number of calorimeter rec-hits carrying 90% of the jet energy)
- fHPD< 0.98 (fraction of jet energy carried by the hottest HPD)[13]

After these cuts, 872 data events and 158440 MC events were selected. The p_T, η and ϕ distributions for the selected leading and sub-leading jet compared to the MC are normalized to the number of data events and shown in figures 5.3, 5.4 and 5.5.

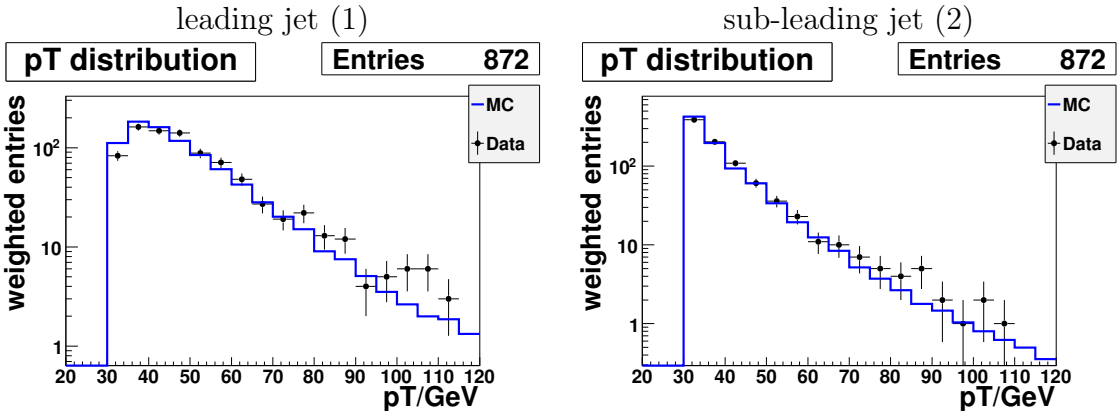

Figure 5.3: The selected Jets: p_T-distribution

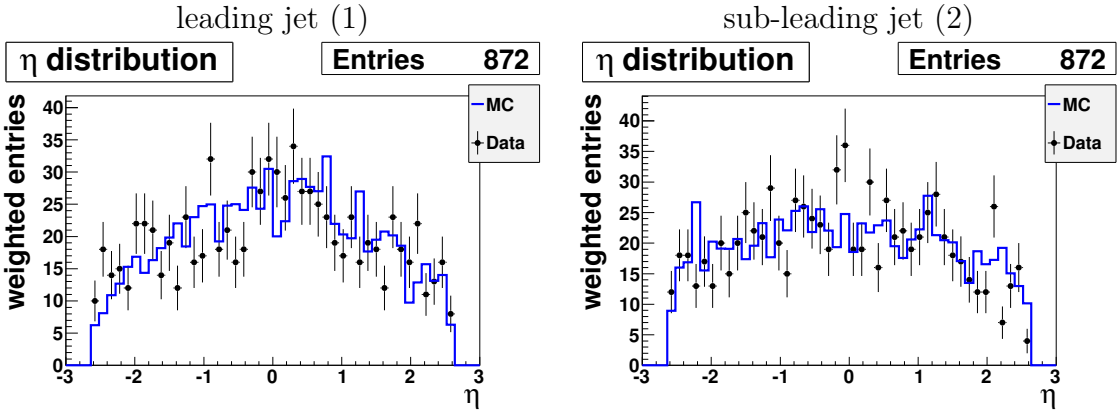

Figure 5.4: The selected Jets: η-distribution

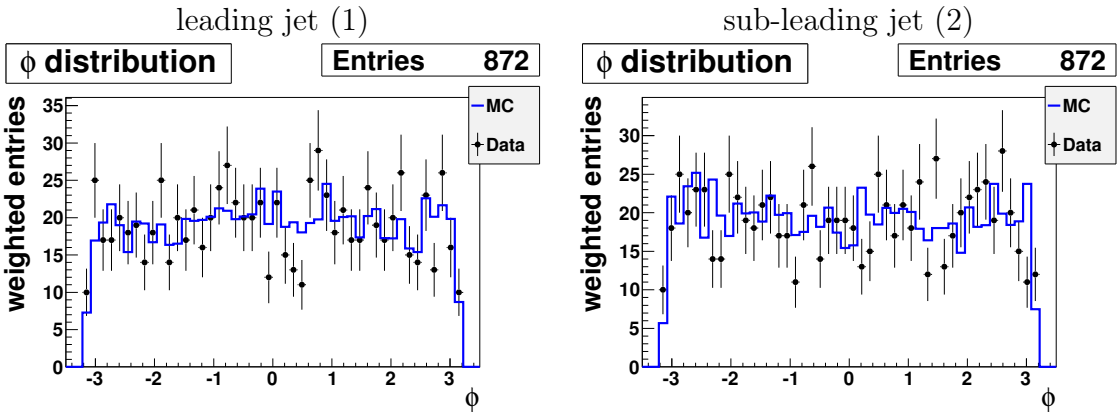

Figure 5.5: The selected Jets: ϕ-distribution

These distributions show that the chosen Monte-Carlo sample is applicable for the data. The dijet mass can be computed by formula 5.1.

$$m_{Dijet} = \sqrt{\left(E_{jet,1} + E_{jet,2}\right)^2 - \left(\vec{p}_{jet,1} + \vec{p}_{jet,2}\right)^2} \tag{5.1}$$

Figure 5.6 shows the result of this analysis which was conducted with the developed program.

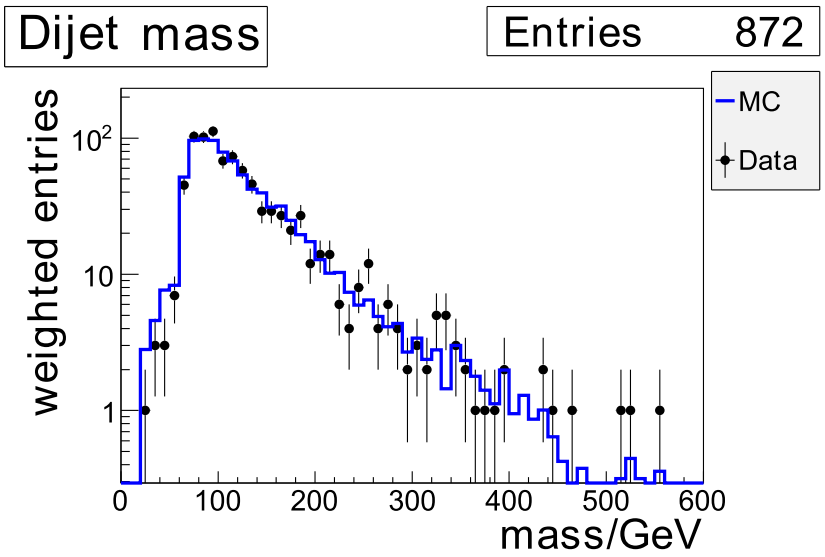

Figure 5.6: The dijet mass

The data is describable by the Monte-Carlo within its uncertainties. This demonstrates that the predicted QCD calculations with Pythia and the detector simulation at 7 TeV is understood within the low statistics.

The performed analysis is shown in figure 5.7.

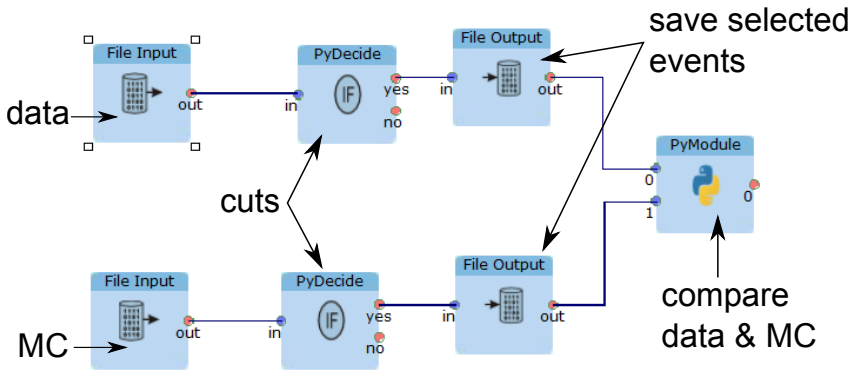

Figure 5.7: The performed analysis inside the web browser

6 Summary

In my thesis, I demonstrated the challenges of the web based physics analysis development environment, 'VISPA@WEB'. I have used various packages like the PXL class collection or JavaScript libraries to fulfill this task. During its implementation, I introduced concepts like the analysis session handling for a parallel operating client management or a multiprocessing job system. I also performed a dijet analysis with recent CMS data to demonstrate its capabilities.

6.1 The new possibilities

The first implementation of 'VISPA@WEB' shows great potential for a new way to perform physics analyses as it has never been before. Its graphical overlay and clarity even represents an abstraction from a physics analysis written in a complex programming language like C++ and redirects the user's focus back to physics. Nevertheless, the underlying PXL class collection assures to handle analyses with any degree of complexity e.g. from a simple $Z^0 \to \mu\mu$ up to a complex single top analysis[27]. Furthermore, the concept supports physicists to work together effectively by exchanging their analyses within an institute or over the web.

Everyone who may be interested and fascinated in prototyping, performing, and verifying a physics analysis is now simply able to do so with his web browser.

List of Figures

List of Tables

List of Abbreviations

AJAX ... Asynchronous JavaScript and XML

ALICE .. A Large Ion Collider Experiment

APD ..... silicon Avalanche PhotoDiode

ATLAS .. A Toroidal LHC ApparatuS

CERN ... European Organization for Nuclear Research

CMS .... Compact Muon Solenoid

CSC ..... Cathode Strip Chamber

DOM .... Document Object Model

DT ..... Drift Chamber

ECAL ... Electromagnetic CALorimeter

GUI ..... Graphical User Interface

HCAL ... Hadron CALorimeter

HEP ..... High Energy Physics

HPD .... multi-channel Hybrid PhotoDiode

HTTP ... Hypertext Transfer Protocol

IO ..... Input and Output

IP ..... Interaction Point

JS ..... JavaScript

JSON ... JavaScript Object Notation

LHC ..... Large Hadron Collider

LHC-b .. Large Hadron Collider beauty Experiment

MC ..... Monte-Carlo

PXL ..... Physics eXtension Library

QCD .... Quantum ChromoDynamics

RFC ..... Request For Comments

RPC ..... Resistive Plate Chamber

RWTH .. Rheinisch-Westfaelische Technische Hochschule Aachen

SHA ..... Secure Hash Algorithm

SM ..... Standard Model

SUSY ... SUperSYmmetry

UHECR . Ultra High Energy Cosmic Ray

UML .... Unified Modeling Language

VISPA .. VISual Physics Analysis

VPT ..... vacuum PhotoTriodes

XML .... Extensible Markup Language

Bibliography

[1] Christoph Berger, *Elementarteilchenphysik*, Jul 2007, 2. Auflage. Springer Berlin.

[2] René Brun, Fons Rademakers, and Suzanne Panacek, *ROOT, an object oriented data analysis framework*, 17 - 30 Sep 2000, 23rd CERN School of Computing,
`http://root.cern.ch`.

[3] James Clark, *The Expat XML Parser*,
`http://expat.sourceforge.net/`.

[4] CMS Collaboration, *CMS Physics Technical Design Report Volume 1: Detector Performance and Software*, 2006, CERN/LHCC, 001.

[5] Microsoft Corporation, *Silverlight*,
`http://www.silverlight.net/`.

[6] National Instruments Corporation, *LabVIEW Development Systems*,
`http://www.ni.com/labview/`.

[7] D. Crockford, *RFC 4627 - JavaScript Object Notation*, Jul 2006,
`http://www.ietf.org/rfc/rfc4627.txt`, `http://www.json.org/`.

[8] Timo Derstappen, *Jamal*,
`http://jamal-mvc.com/`.

[9] Christophe Dolivet, *Edit area*,
`http://sourceforge.net/projects/editarea/`.

[10] M. Erdmann, A. Hinzmann, T. Klimkovich, G. Mueller, and J. Steggemann, *VISPA: Visual Physics Analysis*,
`http://vispa.sourceforge.net/`.

[11] A. Hinzmann et. al., *Visual Physics Analysis - Applications in High-Energy- and Astroparticle-Physics*, 22-27 Feb 2010, 13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research,
`http://acat2010.cern.ch/`.

[12] N. Amapane et al., *Local Muon Reconstruction in the Drift Tube Detectors*, CMS NOTE 2009/008 (2009).

[13] Robert M. Harris et. al., *Event displays of dijet events with highest pT in pp collisions at 900 GeV*, Jan 2010.

[14] C. Amsler et al. (Particle Data Group), *Review of Particle Physics. Physics Letters B667*, 1 (2008).

[15] Communication Group, *CERN-Brochure-2009-003-Eng*, Feb 2009, http://cdsmedia.cern.ch/img/CERN-Brochure-2009-003-Eng.pdf.

[16] Chuck Hagenbuch, Jan Schneider, and Michael Slusarz, *DIMP*, http://www.horde.org/dimp/.

[17] Andreas Herz, *Open-jACOB Draw 2D*, http://www.draw2d.org/.

[18] Ian Hickson and David Hyatt, *A vocabulary and associated APIs for HTML and XHTML*, Aug 2009, http://www.w3.org/TR/2009/WD-html5-20090825/.

[19] Adobe Systems Inc., *Flash*, http://www.adobe.com/products/flash/.

[20] Sencha Inc., *Ext JS, Cross-Browser Rich Internet Application Framework*, http://www.sencha.com/products/js/.

[21] Garrett J. J, *AJAX: A New Approach to Web Applications*, Feb 2005, http://www.adaptivepath.com/ideas/essays/archives/000385.php.

[22] JLB.PP, *CMS PARA 001*, 1997, http://www-collider.physics.ucla.edu/cms/.

[23] Klaus Kämpf, *SWIG - Generating language bindings for C/C++ libraries*, Jul 2008, http://www.swig.org.

[24] Eric Knorr, *2004 - The Year of Web Services*, Fast-Forward 2010 - The Fate of IT (2003).

[25] Christiane Lefèvre, *The CERN accelerator complex*, Dec 2008, http://cdsweb.cern.ch/record/1260465.

[26] Sun Microsystems, *Java-Applets*, http://java.sun.com/applets/.

[27] Gero Müller, *Development and Application of a Novel Graphical Environment for Physics Data Analysis with the CMS Experiment*, Sep 2008, Diplomarbeit, RWTH Aachen.

[28] Valerio Proietti, *MooTools - My Object Oriented JavaScript Tools*, http://mootools.net/.

[29] Greg Roelofs and Mark Adler, *ZLIB, A Massively Spiffy Yet Delicately Unobtrusive Compression Library*, http://zlib.net/.

[30] J. Steggemann, T. Winchen, G. Mueller, and M. Erdmann, *PXL: Physics extension library*, http://pxl.sourceforge.net/.

[31] Walter Zorn, *JavaScript Vector Graphics library.*

Danksagung

Ich möchte mich bei dem gesamten VISPA Team, für die gute Zusammenarbeit, für Tipps und Ideen die das Gelingen dieser Arbeit erst ermöglicht haben bedanken. Für die Hilfe bei der Analyse danke ich Oliver Delpy und Andreas Hinzmann.

Besonders danke ich Prof. Martin Erdmann und Gero Müller für die absolut exzellente Betreuung.

Erklärung

Hiermit versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe.

Aachen, den 20. Juli 2010 _____