Journal of Physics: Conference Series **219** (2010) 042041 doi:10.1088/1742-6596/219/4/042041

# Visual Physics Analysis VISPA

**Oxana Actis, Michael Brodski, Martin Erdmann, Robert Fischer, Andreas Hinzmann, Tatsiana Klimkovich[1], Gero Müller, Thomas Münzer, Matthias Plum, Jan Steggemann, Tobias Winchen**

Physics Institute 3a, RWTH Aachen University, 52056 Aachen, Germany

E-mail: tatsiana.klimkovich@physik.rwth-aachen.de

**Abstract.** VISPA is a development environment for high energy physics analyses which enables physicists to combine graphical and textual work. A physics analysis cycle consists of prototyping, performing, and verifying the analysis. The main feature of VISPA is a multipurpose window for visual steering of analysis steps, creation of analysis templates, and browsing physics event data at different steps of an analysis. VISPA follows an experiment-independent approach and incorporates various tools for steering and controlling required in a typical analysis. Connection to different frameworks of high energy physics experiments is achieved by using different types of interfaces. We present the look-and-feel for an example physics analysis at the LHC and explain the underlying software concepts of VISPA.

## 1. Introduction

In the past decade, major progress has been achieved in the development of experiment-specific software analysis frameworks, typically based on C++ and a specific configuration language (e.g. [1, 2]). In some of them the dynamically-typed programming language Python is being used for the analysis environment [3, 4]. The visualizations of a physics analysis has been attempted in the past [5] and has recently become a field of strong interest again. With the project VISPA (Visual Physics Analysis) [6, 7], the step in between visualizing the measured objects in the detector (event display) and physics distributions (histograms) is graphically supported. In this step, the primary tasks of a physicist are prototyping, executing, and verifying a physics analysis (Fig. 1). This is an iterative procedure until the analysis is finalized.

VISPA facilitates the analysis cycle by combining graphical and textual programming. This combination has shown to speed up design and development in other fields, e.g. hardware control using the LabView program [8].

To deploy a visual environment for physics analysis, VISPA provides a multi-purpose window tool with a three column structure - a navigator panel, a window for graphical displays, and a property panel (Fig. 2). For the text-based programming, both the C++ and the Python languages are supported. VISPA has been developed independently of experiment-specific software. It has well defined interfaces to connect to any high energy physics experiment. The C++ toolkit PXL [9] serves as an underlying analysis software for the VISPA framework.

For designing a physics analysis, VISPA provides a graphical module steering which enables physicists to add, connect, and configure the analysis modules. Based on a plug-in mechanism,

[1] Presenter

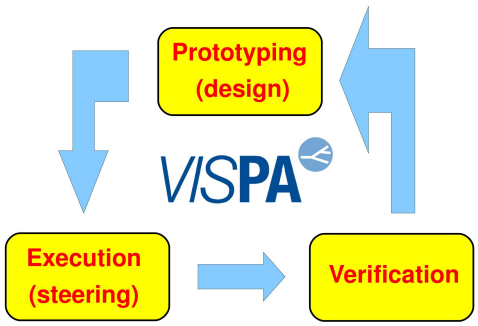


**Figure 1.** Analysis steps provided by VISPA.

modules are already provided for the purpose of reading/writing event data, or for individual user analysis code. The latter can be coded directly within a corresponding editor. The visualization of the module steering facilitates verification of the analysis structure and communication between physicists doing analysis (Fig. 2). For verification of the analysis, VISPA provides an event browser to inspect the overall event structure, to display particle decay cascades, and the properties of each particle (Fig. 3).

This contribution is organized as follows: first, the underlying C++ toolkit PXL is described. Then, the Graphical User Interface of VISPA and the Event Browser will be discussed. After this, the design of the analysis within the VISPA framework will be explained. Finally, further developments will be mentioned and different types of interfaces exploited by VISPA will be discussed.

## 2. The PXL Toolkit

The C++ toolkit PXL (Physics eXtension Library) has been developed since 2006 [9]. It is the successor of the PAX toolkit, which was developed from 2002 to 2007 [10, 11]. In 2009 the version 2.1 of the PXL toolkit has been released.

PXL provides all necessary features for an experiment-independent high level physics analysis with emphasis on user-friendly programming syntax. Particularly, PXL enables the reconstruction of decay trees and the handling of analyses with reconstruction ambiguities. PXL offers an extensible collection of physics objects, representing particles (*pxl::Particle*), vertices (*pxl::Vertex*) and collisions (*pxl::Collision*). In the analysis of an event containing reconstructed data, new information can be added to each object by means of user event data (*pxl::UserRecord*). Between all objects, relations can be established, e.g. to build up decay trees, or to associate reconstructed particles with generated particles. The class *pxl::Event* represents an entire physics event and *pxl::EventView* is a special view of this physics event. These classes act as containers for physics objects, holding the relations between them, and serve as the standard interface to algorithms. Copies of these class instances preserve all contained information such as the relations between particles. These features efficiently support the evaluation of possible reconstructed particle cascades.

All objects included into the PXL I/O scheme inherit from *pxl::Serializable*, e.g. an event container *pxl::Event* with its physics objects and relations between them. The PXL I/O scheme can handle the user defined classes which are explicitly included into it. Each file is made up of isolated compressed binary data chunks which e.g. contain a certain number of events. The current I/O system is able to handle the files from a local disk. In the future handling files of the dCache system will be included.

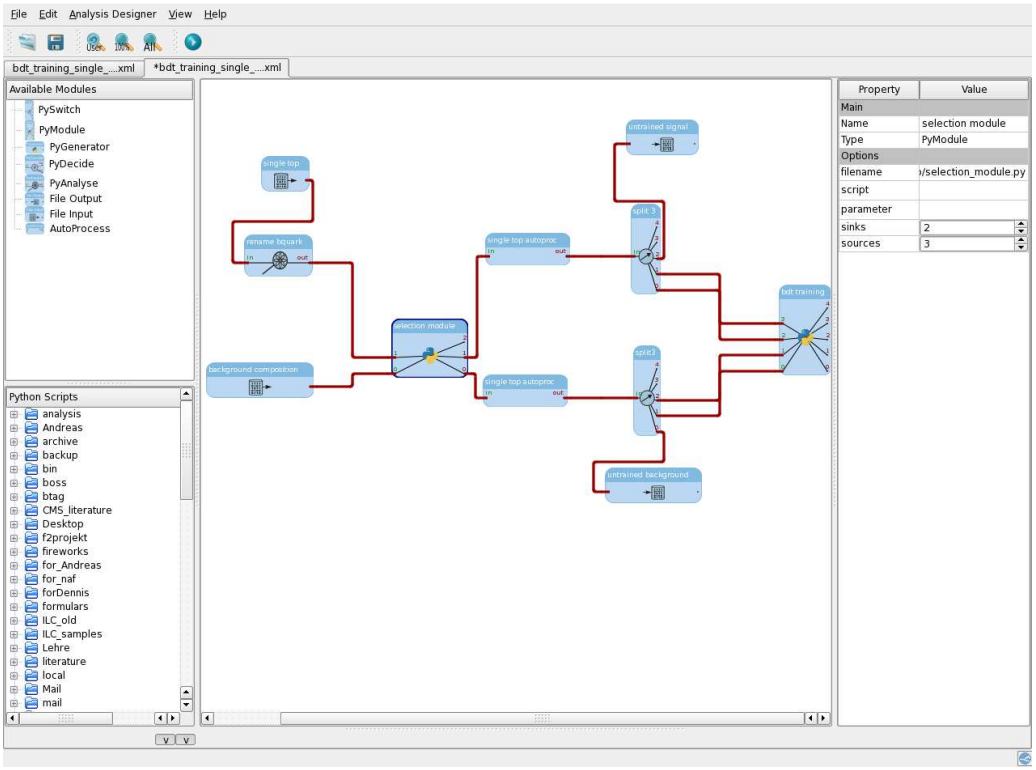


**Figure 2.** Analysis design within the VISPA graphical platform.

To enable the usage of all PXL objects and their methods within Python programs, a Python extension PyPXL is provided (see Section 5.4).

The structure of the PXL toolkit is the following. The core packages are the *base* library and the *io* library. The *base* library contains classes for relation management, user data handling, basic containers, etc. The *io* library has classes for input/output. The *hep* library contains more specific classes for high energy physics analysis. For astroparticle physics experiments the *astro* library is provided. The packages *plugins* and *modules* are especially designed for VISPA (see Sections 3 and 5.3).

## 3. Graphical User Interface (GUI) of VISPA

The VISPA multi-purpose window serves as the graphical user interface for the design and steering of analyses, and for the browsing of complete physics events. It provides a common user interface for these different tasks (Figs. 2, 3).

The Graphical User Interface of VISPA is designed as a framework for the applications for graphical browsing and editing. It has been developed for maximum reusability. The browsers and editors are plug-ins of the main application which controls the main window. The main window provides icons for opening and saving data files. By using a tabbed document interface, several files can be opened in parallel.

The plug-in browsers or editors define the content of the main window. Typically the main window is split into three views. The graphical compounds, such as views or widgets, are shared between different plug-ins. For example, the right view is typically a property grid which lists the properties of an object selected in the center view.

The GUI framework is written in the Python programming language for fast development

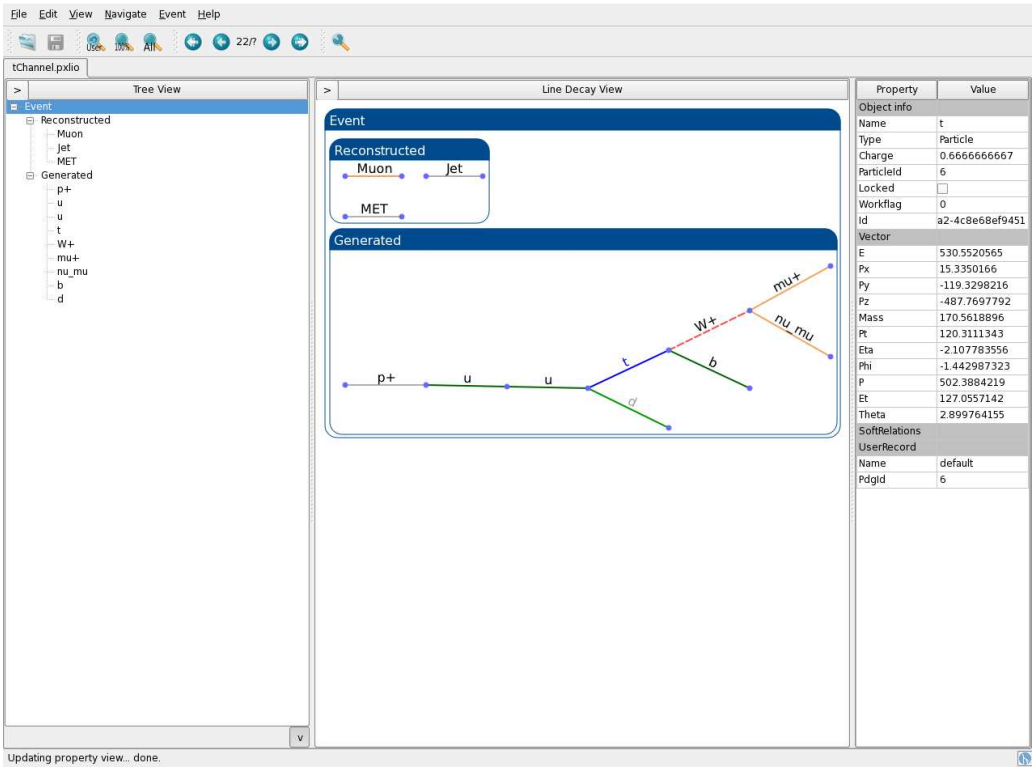


**Figure 3.** The Event Browser within the VISPA graphical platform.

turnaround due to dynamic typing and automatic memory management. It is based on Qt [12], a widely spread cross-platform application and user interface development framework. Due to its implementation in PyQt [13], VISPA has been successfully distributed on Linux, Mac OS X and Windows based systems.

## 4. The Event Browser

At the beginning of any analysis it is often useful to inspect the input data. VISPA provides such a tool not only for the input data, but also for inspecting the structure and the contents of the data at any step of the analysis. The Event Browser is designed as a plug-in of the VISPA GUI (see Section 3). Browsing physics event data in VISPA allows the verification of physics analyses on an event-by-event basis (Fig. 3). In the left window a complete content of an event is represented as a tree. Each object can be explored by selecting it and inspecting its contents in the property grid on the right-hand side. The user can search or filter objects with respect to their name, any of their properties, or using a user defined Python script. In addition, the visualization of decay trees allows to check if all relations have been established correctly. In the decay tree the particle type (boson, lepton, quark) is indicated by its color and line shape.

To display complex decay trees, the VISPA Event Browser incorporates an algorithm for their proper layout (*pxl::AutoLayout*). The algorithm is based on a model of physical forces, like spring forces, or gravity. Each starting point and end point of a particle is provided with a node subjected to these forces. Using an iterative procedure, the positions of the nodes are optimized with respect to balanced forces. This algorithm results in a well-distributed view of, e.g., asymmetric decay trees.

## 5. VISPA Components for Designing a Physics Analysis

In this section, the software tools provided by the VISPA environment for prototyping the analysis will be explained.

### *5.1. Analysis Designer*

The Analysis Designer is developed as a plug-in of the VISPA GUI (see Section 3). The content of the main window is the following (Fig. 2): the frame on the left-hand side contains the list of available modules, the main window displays the analysis modules, and the frame on the right-hand side shows the properties of the item selected in the main window. All objects in the main frame can be selected and moved like in popular software (drag-and-drop). Icons for opening and saving data files as well as other analysis modules are provided. It is possible with double click on, e.g., the input file module to open the data file.

### *5.2. Module Steering*

The design of a physics analysis with VISPA is based on the decomposition of an analysis into modules. Whereas a simple analysis may require only a few modules which are connected serially, a complex analysis requires more modules and more sophisticated streams of data. The VISPA module steering system controls the data flow as well as the selection and settings of analysis modules. The module selection is based on a plug-in mechanism, guaranteeing extensibility and efficiency as only libraries for the analysis modules deployed in the current analysis need to be provided. The data flow is managed by connections between sinks (data input) and sources (data output) of the respective modules. This permits the usage of multiple data streams as needed in complex physics analyses.

### *5.3. Analysis Modules*

A variety of analysis modules is provided for tasks of different complexity within a physics analysis. File operations are handled by input and output file modules. For user-demanded tasks within an analysis, a number of Python scriptable modules are provided, such as event generator *PyGenerator*, decision module *PyDecide* etc.

One of the available modules is the C++ based analysis module *AutoProcess*. It is developed for the automated reconstruction of particle cascades, a task arising in the reconstruction of particles with combinatorial ambiguities [14]. Given a template of a particle decay cascade and reconstructed particle data as input, this module generates all possible reconstruction versions. For events simulated with a Monte Carlo generator it supports finding the reconstructed version corresponding to the correct decay chain. The performance of the module is optimized for low memory and CPU time consumption.

Other user modules can be written in C++ and steered within VISPA module steering system.

### *5.4. Python Interface*

To enable the usage of all PXL objects and their methods within Python programs, a Python extension PyPXL is provided. A fragment of Python code using PyPXL for jet selection looks as follows:

```
for particle in eventview.getParticles():
   if (particle.getName() == 'Jet' and particle.getPt() > 30.):
     print 'Jet with Pt > 30.'
```

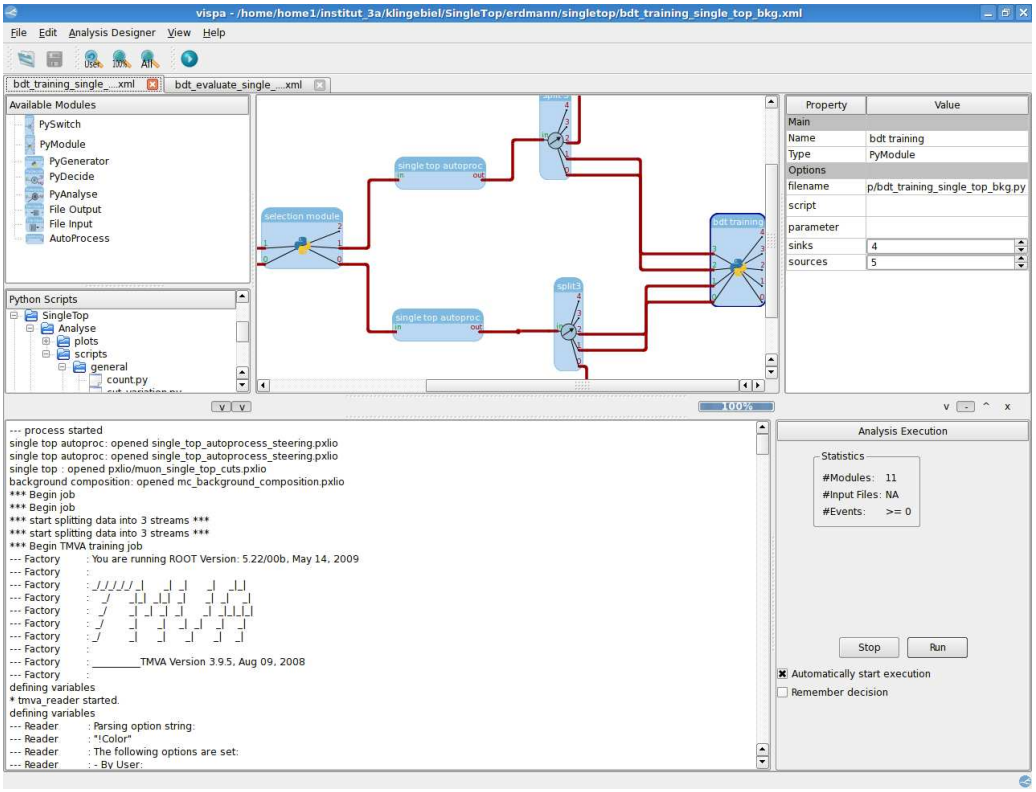


**Figure 4.** Executing the analysis interactively within VISPA.

## 6. Analysis Flow in VISPA

A typical analysis flow within the VISPA framework is demonstrated in Fig. 1. In a first step, a physicist performs an inspection of the input data using the Event Browser (see Section 4). After this, the design of the analysis chain using the Analysis Designer (see Section 5.1) takes place. A user develops an analysis script in the Python programming language. Any state of the analysis design can be stored and received back either in XML format or in Python. The analysis can be executed either in batch mode, or interactively from the VISPA graphical window (Fig. 4).

After the execution of the analysis its results can be verified using ROOT [15, 16] histograms. The output of the analysis modules for particle reconstruction can be checked in the Event Browser. As an example, the output of the automated reconstruction of particle cascades for the case of single top production is demonstrated in Fig. 5.

## 7. Further Developments and Use

In the CMS experiment the VISPA GUI (see Section 3) is used as a platform for the Configuration Browser [17, 18]. This application allows to visually browse and edit the job configuration system of the CMS experiment.

A further plug-in project which is under development using the VISPA GUI is the Edm Browser [19] - an event browser which allows to inspect the content of CMS data files, event by event.

Currently the VISPA framework is used for different CMS analyses, e.g. a single top analysis [20], a model independent search, etc. Currently, an interface between the ILC format and the PXL format is being developed. The framework is also being used for student exercises within the course “Elementary particle physics” at the RWTH Aachen University.

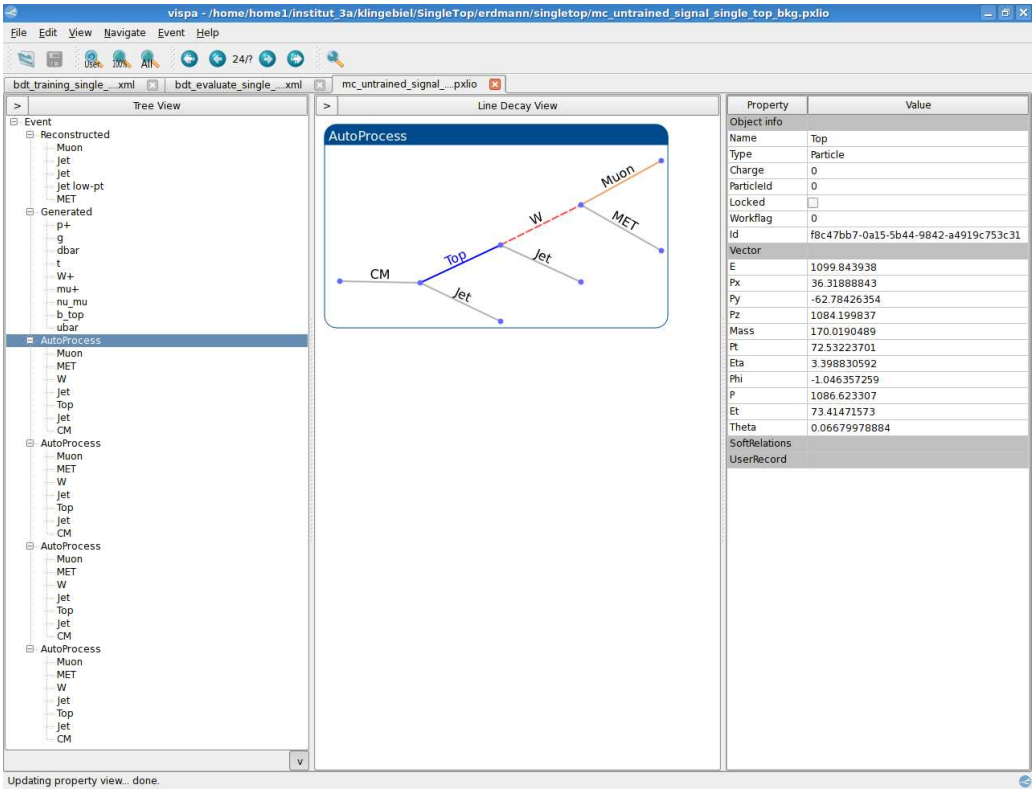


**Figure 5.** Verification of analysis results within VISPA: output of the *AutoProcess* module.

## 8. Connection to Experiments

There are different types of interfaces exploited by VISPA to connect to any high energy physics experiment.

The possibility of performing analyses in any experiment is provided by an interface based on the C++ toolkit PXL (Section 2). Any experiment specific data can be stored in the experiment independent PXL I/O format using the pxl:Event, a general container to hold all relevant information of a high energy physics event.

The Python interface to the software classes from different experiments allows to use VISPA graphical tools in combination with data from any experiment, by means of introspection. For instance, VISPA accesses PXL classes through PyPXL. The Edm Browser from CMS inspects in this manner the content of CMS data files using the PyEDM.

The third interface to experiments provided by VISPA is the GUI framework of VISPA which allows to create graphical browsing and editing applications as plug-ins of the main application. The examples of such applications are the Event Browser and the Analysis Designer in VISPA, the Configuration Browser and the Edm Browser at CMS.

## 9. Support

All the software is continuously maintained, fully documented and available online [7, 9]. Recently the version 0.1.3 of VISPA and the version 2.1.3 of PXL have been released. There are installers provided for Windows, Mac OS X and Debian/Ubuntu platforms.

## 10. Conclusion

A physics analysis environment, VISPA, has been presented. VISPA facilitates prototyping, performing, and verifying a physics data analysis by combining graphical and textual programming. It provides the so far missing graphical support for physicists in the step between displaying events, and visualizing physics distributions.

## 11. Acknowledgements

We are very grateful for financial support of the Ministerium für Innovation,Wissenschaft, Forschung und Technologie des Landes Nordrhein-Westfalen, the Bundesministerium für Bildung und Forschung (BMBF), the Deutsche Forschungsgemeinschaft (DFG) and Helmholtz Alliance "Physics at the Terascale".

## References

[1] A. B. Meyer [H1 Collaboration], A new object-oriented physics analysis framework for the H1 experiment, Prepared for *International Europhysics Conference on High-Energy Physics (HEP 2001)*, Budapest, Hungary, 12-18 Jul 2001.
[2] F. Fabozzi, C. D. Jones, B. Hegner and L. Lista, Physics analysis tools for the CMS experiment at LHC, CERN-CMS-NOTE-2008-015.
[3] G. Barrand, M. Frank, P. Mato, E. de Oliveira, A. Tsaregorodtsev and I. Belyaev, Python-based physics analysis environment for LHCb, In *Interlaken 2004, Computing in high energy physics and nuclear physics*, 377-380.
[4] C. D. Jones, L. Luca and B. Hegner, Analysis Environments for CMS, *J. Phys. Conf. Ser.* **119** (2008) 032027.
[5] B. Ferrero Merlino, The LHC++ environment, CERN-OPEN-2000-191.
[6] O. Actis *et al.*, Visual Physics Analysis (VISPA) - Concepts and First Applications, arXiv:0810.3609 [physics.data-an]
[7] VISPA (Visual Physics Analysis), http://vispa.sourceforge.net
[8] National Instruments, LabView, http://www.ni.com/labview
[9] M. Erdmann, G. Mueller, J. Steggemann, Physics eXtension Library 2.0, http://pxl.sourceforge.net
[10] M. Erdmann, D. Hirschbuehl, Y. Kemp, P. Schemitz and T. Walter, User oriented design in high energy physics applications: Physics analysis expert, Prepared for *14th Topical Conference on Hadron Collider Physics (HCP 2002)*, Karlsruhe, Germany, 29 Sep - 4 Oct 2002
[11] S. Kappler *et al.*, The PAX toolkit and its applications at Tevatron and LHC, *IEEE Trans. Nucl. Sci.* **53** (2006) 506 [arXiv:physics/0512232]
[12] Qt - A cross-platform application and UI framework, http://www.qtsoftware.com
[13] PyQt, http://www.riverbankcomputing.co.uk/software/pyqt
[14] O. Actis *et al.*, Automated Reconstruction of Particle Cascades in High Energy Physics Experiments, arXiv:0801.1302 [physics.data-an]
[15] R. Brun and F. Rademakers, ROOT: An object oriented data analysis framework, *Nucl. Instrum. Meth.* A **389** (1997) 81
[16] ROOT, http://root.cern.ch
[17] Configuration Browser for CMS, https://twiki.cern.ch/twiki//bin/view/CMS/SWGuideConfigBrowser
[18] M. Erdmann, R. Fischer, B. Hegner, A. Hinzmann, T. Klimkovich, G. Müller, J. Steggemann, Visualisation of the CMS Python Configuration System, Prepared for *17th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2009)*, Prague, Czech Republic, 21 - 27 March 2009
[19] Edm Browser for CMS, https://twiki.cern.ch/twiki//bin/view/CMS/SWGuideEdmBrowser
[20] G. Müller Development and Application of a Novel Graphical Environment for Physics Data Analysis with the CMS Experiment, Diploma thesis, September 2008