Bringing Experiment Software to the Web with VISPA

Bringing Experiment Software to the Web with VISPA

M Erdmann, B Fischer, R Fischer, C Glaser, F Heidemann, G Müller, T Quast, M Rieger, M Urban, D van Asseldonk, R F von Cube and C Welling

III. Physics Institute A, RWTH Aachen University, Germany

E-mail: rieger@physik.rwth-aachen.de

Abstract. The Visual Physics Analysis (VISPA) software is a toolbox for accessing analysis software via the web. It is based on latest web technologies and provides a powerful extension mechanism that enables to interface a wide range of applications. It especially meets the demands of sophisticated experiment-specific use cases that focus on physics data analyses and typically require a high degree of interactivity. As an example, we developed a data inspector which is capable of browsing interactively through event content of several data formats, e.g., MiniAOD which is utilized by the CMS collaboration. Visual control of a chain of user analysis modules as well as visualization of user specific workflows support users in rather complex analyses at the level of $t\bar{t}H$ cross section measurements. The VISPA extension mechanism is also used to embed external web-based applications which benefit from dynamic allocation of user-defined computing resources via SSH. For example, by wrapping the JSROOT project, ROOT files located on any remote machine can be inspected directly through a VISPA server instance. We present the techniques of the extension mechanism and corresponding applications.

1. Introduction

Scientific workflows can strongly benefit from the advantages of web applications. Especially location-independent access via web browsers and centralized software installation on dedicated servers are convincing arguments. As an example, *IPython Notebooks* [1] become more and more popular in the physics community.

However, specialized experiment-specific software requires an entirely different level of infrastructure in order to profit from web-based approaches. The VISPA system is capable of coping with the demands of complex experiment environments in the web [2].

The article is structured as follows. Section 2 describes the architecture of the VISPA software as well as its key concepts. Subsequent sections 3 to 5 present developed extensions that demonstrate its suitability as an enviroment for experiment-specific software.

2. The VISPA Software

Following a server-client approach, the VISPA software consists of two parts. The server is written in Python and is implemented using the *CherryPy* framework [3]. The browser-based client is written in JavaScript with graphical components defined in HTML5 [4] and CSS3 [5]. Communication between the two relies on *Hypertext Transfer Protocol* (HTTP) requests and *WebSockets*. User-related information, i.e. login data, permissions and preferences, are

stored in a relational database which is accessed via *object-relational mapping* (ORM) using the *SQLAlchemy* toolkit [6]. The internal user management system enables administrators to group users into logical entities, such as *Projects* and *Groups*, and handles allocation of permissions.

Two key concepts are the ability to connect to user-defined resources, called *workspaces*, and a mechnism that allows for extending functionalities. Both of them are described in detail in the following.

2.1. Workspace Architecture

The purpose of the VISPA server is to provide a working environment for multiple users. Thus, in order to preserve server responsiveness, requests for large amounts of data and computing power should be treated separately. Workspaces constitute the means to including dynamic and scalable resources into the server-client architecture. A schematic overview is shown in figure 1.

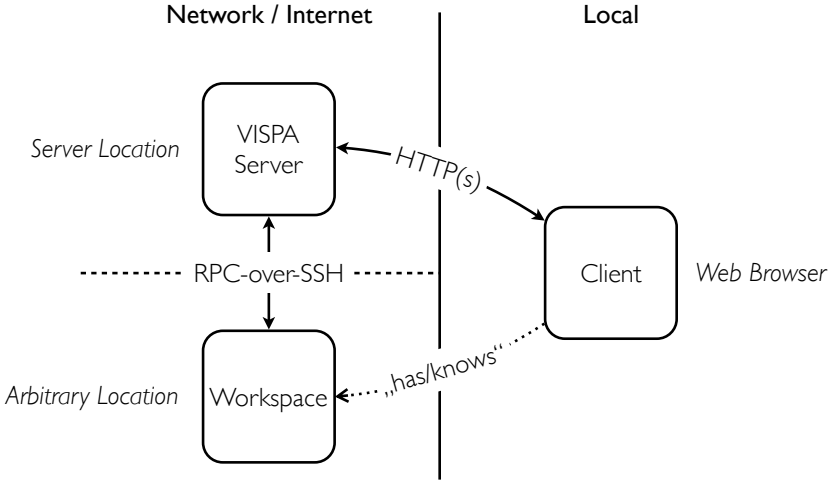

Figure 1: Schematic overview of the workspace architecture [7].

A workspace is a worker node that runs a Python interpreter and is accessible through a network via SSH. These requirements can be met by numerous different machines: desktop computers, computing clusters, and even tablets and smartphones. Resource utilization and file access are granted as the server relies on the user login to interact with workspaces via *remote procedure calls* (RPC). The connection between server and clients as well as between server and workspaces is encrypted via SSL and SSH, respectively.

Using the graphical interface of the client, users can connect to predefined workspaces and, unless explicitly forbidden by an administrator, define and connect to custom workspaces.

2.2. Extension Mechanism

Usually, experiment-specific software is specially tailored for its particular use case. Any system or environment that aims to cover several use cases simultaneously should exhibit appropriate flexibility. Within VISPA, this is achieved by separating core functionality from implementation of specific features using an extension machanism.

Extensions are installed on the machine that runs the VISPA server instance and are typically available for every user that can access the server. They are composed of three parts: code and graphics markup that extend the client (JavaScript/HTML/CSS), code that handles the communication on the server (Python), and, optionally, code that is transferred to and executed on workspaces (Python). The purpose of the latter part is to outsource and encapsulate actual extension logic causing the server to take on the role of a mediator between requests and resources.

Typical basic extensions are file browser, file selector, code editor, and terminal. As they are essential for most common workflows, they are pre-installed on the VISPA server. But due to the vast development of web techniques over the past years as well as the abundant supply of existing web-based software packages, even more complex and specialized extensions become feasible. Possible applications range from visualization of static data to highly interactive interfaces, such as graphical steering of modular analyses which is presented below.

3. Analysis Designer and Data Browser

The extensions presented in this section rely on the *Physics eXtension Library* (PXL), which is a collection of C++ classes addressing analyses in high energy physics (HEP) and astroparticle physics [8]. It features elementary objects such as `pxl::LorentzVector` or `pxl::Event` classes, as well as a module system for structuring complex analysis code into reusable building blocks. Modules can be written in both, C++ and Python. Objects stored in *pxlio* files can be nested and connected to each other via *ownership* and *relations*. They also have the ability to store *user records*, i.e., key-value pairs of arbitrary data types. Therefore, PXL provides a consistent interface to objects and containers which can be organized in various experiment-specific setups.

Chains of analysis modules can be created graphically using the *Analysis Designer* (fig. 2a). Users can add, remove and arrange modules using drag and drop gestures. Options give rise to code flexibility beyond compile time. Connections between modules control the data flow during execution initiated via the `pxlrun` command.

Input and output files of these analysis chains can be inspected by the *Data Browser* (fig. 2b). In the case of HEP data, browsing is performed on a per-event basis. Different views on the event and its contents are presented in dedicated windows. Currently, Feynman-like graphs, plain content listings and detailed property listings are supported. However, the code structure allows for the addition of further views such as, e.g., a three-dimensional event display. Based on the non-restrictive design of PXL objects as explained above, event content stored in other data formats is convertible to *pxlio* and thus, visualizable by the extension.

4. Analysis Workflow Management and Visualization

For mastering a specific research question, management of underlying scientic workflows poses a complex challenge in today's physics working environments. Current HEP analyses utilize a significant amount of resources in terms of CPU time, memory consumption, and disk space. The logical solution to resource requirements is parallelization on large-scale computing centers. Main challenges are, e.g., ten thousands of datasets to be processed, application of multivariate analysis concepts with execution dependencies for training and evaluation data, and the complexity of reconstruction and analysis algorithms. However, physicists are usually required to steer their individual workloads manually, i.e., start (remote) processes, monitor their execution, handle potential failures, and eventually obtain intermediate output data to be passed to further analysis. In addition, all *dependent* workloads have to be initiated once the current one finished successfully. On a large scale, this *management* task is not only time-consuming for the operating physicist, but, in contrast to automated approaches, also represents a risk for errors, e.g., the loss of information on the interplay between particular workloads. In fact, in certain cases the replicability of physics results might not be guaranteed.

We developed the Analysis Workflow Management tool (AWM) which aims at reducing the complexity described above. It provides guidance on structuring arbitrary workloads, so-called *steps*, into directional workflows. Dependencies are resolved automatically by evaluating the dataflow, i.e., data input and output, between particular steps. Scalability is ensured by introducing resources as abstract locations. *Storage locations*, e.g. local hard-drives or remote data centers, define where data can be stored to and accessed from. *Run locations*, e.g. local computers or computing clusters, describe where data is processed. Along with an internal

bookkeeping system, AWM provides a high degree of automation of typically overhead tasks like job creation and submission, or file transfers. The information exchanged between steps is defined by the user. An integrated software managing system allows for custom user-installed software including installation of required dependencies on run locations. It is written in Python and allows for embedding any executable in customized steps.

In order to display relations of the steps that constitute a workflow, we integrated a *Workflow Visualizer* as an extension for VISPA. It provides a novel approach of illustrating the structure of individual physics analyses (fig. 2c). Graphs are generated by traversing a workflow, collecting information on inputs and outputs of steps to infer relations, and passing them encoded as *dot* files to algorithms of the *graphviz* package [9]. The resulting *scalable vector graphics* (svg) is then displayed in the web browser. In case of complex workflows, users can interactively change the appearance of a graph by combining multiple steps refering to similar tasks into so-called *clusters*. It is also possible to query detailed information about each step.

5. Embedding External Functionality: JSROOT

The number of portings of successful desktop software projects to the web increased rapidly during the last years. A physics-related example is the ROOT framework [10]. In ROOT, one and two dimensional histograms as well as flat datastructures can be visualized with a high degree of interactivity. This particular feature, called `TBrowser`, was ported to the web using JavaScript and HTML, and released as JSROOT. It is even capable of rendering three dimensional objects using *WebGL*. In order to visualize data, software installation and target files have to be located on the same machine. In certain cases, target files need to be copied to the software location or vice versa.

As explained in section 2, VISPA simplifies remote data access using workspaces while requiring only one software installation on a single VISPA server instance. Technically, JSROOT can be used as a plugin. We developed a VISPA extension consisting of only a thin code layer that embeds this plugin which now benefits from exchangeable workspaces (fig. 2d). It supplies the same features and rich interactions as the standalone JSROOT application.

6. Conclusions

The VISPA project provides an ecosystem for experiment-specific software. Its architecture can be considered a "glue" between multiple web-based applications and combines them into a single, customizable environment to exploit synergies. Custom extensions have been developed to address several use cases for physics analyses.

The event content of HEP data files can be interactively visualized with the *Data Browser*. The *Analysis Designer* provides a graphical interface to modular analysis chains based on PXL. For the purpose of coping with the complexity of modern physics analyses, an *Analysis Workflow Manager* has been developed. This tool allows physicists to define analysis steps, structure them within workflows, and control their execution across arbitrary locations. These workflows can be displayed using a dedicated VISPA extension. With the example of JSROOT, we showed that the extension mechanism is also capable of embedding existing software. Executing JSROOT through VISPA enhances flexibility as it benefits from the concept of workspaces.

Acknowledgments

We wish to thank the conference organizers for their kind support. This VISPA project is supported by the Ministerium für Wissenschaft und Forschung, Nordrhein-Westfalen, the Bundesministerium für Bildung und Forschung (BMBF), the Helmholz Alliance Physics at the Terascale, and the Deutsche Forschungsgemeinschaft.

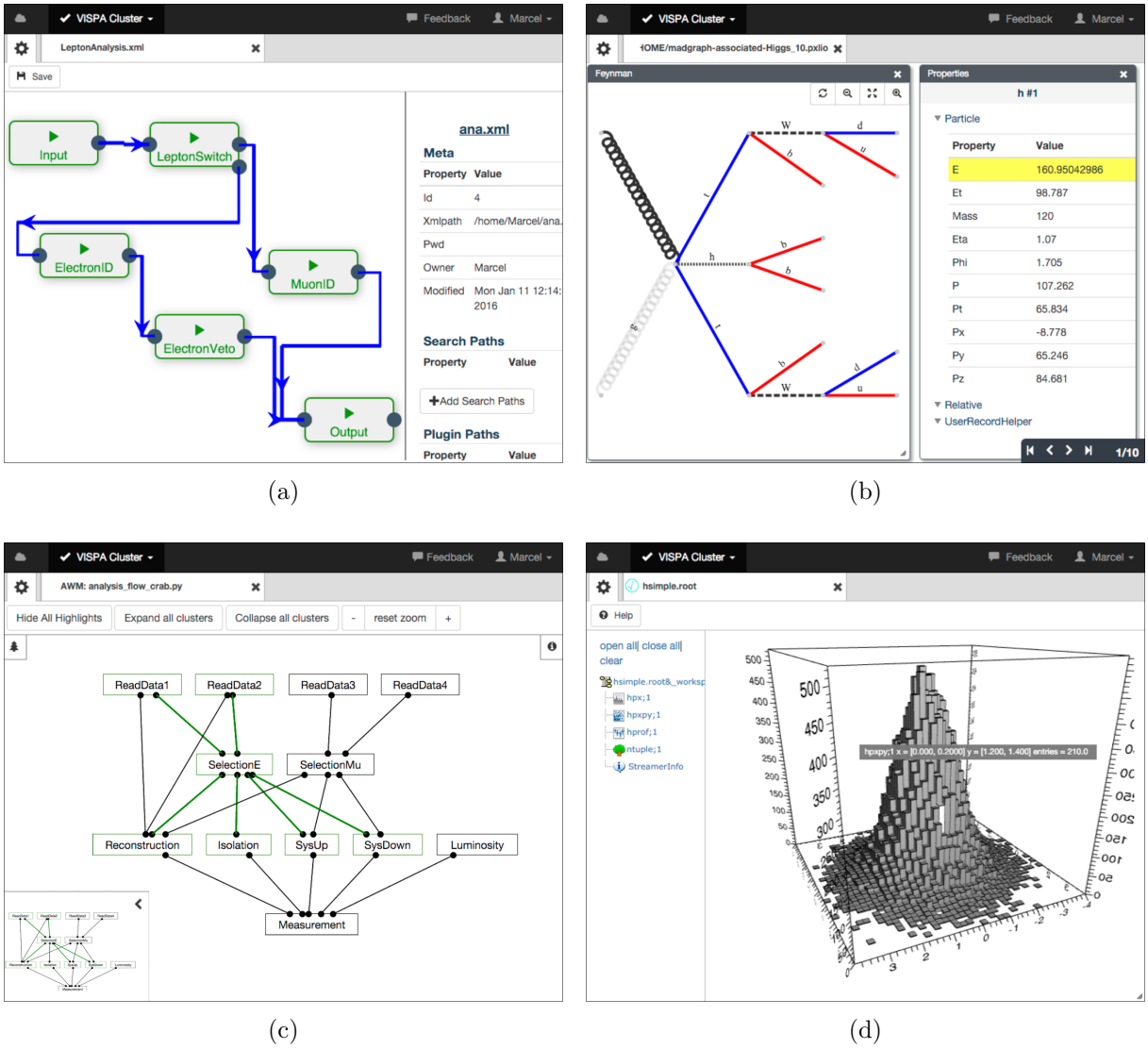

Figure 2: Analysis-specific extensions of the web-based implementation of VISPA. a) Analysis Designer, b) Data Browser, d) Workflow Visualizer, and d) JSROOT.

References

[1] Fernando P and Granger, B E. (2007), *IPython: a System for Interactive Scientific Computing*, IEEE Comput. Sci. Eng., vol. 9, no. 3, pp21-29, doi:10.1109/MCSE.2007.53, `http://ipython.org`

[2] Bretz H-P et al. (2012), *A Development Environment for Visual Physics Analysis*, JINST **7** T08005, doi:10.1088/1748-0221/7/08/T08005, arXiv:1205.4912v2, `http://vispa.physik.rwth-aachen.de`

[3] CherryPy, `http://www.cherrypy.org`

[4] World Wide Web Consortium, *HTML5 Specification*, `http://www.w3.org/TR/html5`

[5] World Wide Web Consortium, *CSS3 Specification*, `https://www.w3.org/Style/CSS`

[6] SQLAlchemy, `http://www.sqlalchemy.org`

[7] Erdmann M et al. (2014), *A Web-Based Development Environment for Collaborative Data Analysis*, J. Phys.: Conf. Ser. **523** 012021, doi:10.1088/1742-6596/523/1/012021

[8] Erdmann M et al. (2012), *Physics eXtension Library (PXL)*, `http://vispa.physik.rwth-aachen.de/pxl`

[9] Gansner E R and North S C (2000), *An open graph visualization system and its applications to software engineering*, Software Pract Exper, vol. 30, no. 11, pp1203-1233, `http://www.graphviz.org`

[10] Brun R and Rademakers F (1996), *ROOT - An Object Oriented Data Analysis Framework*, Nucl. Inst. & Meth. in Phys. Res. A 398 81-86, `http://root.cern.ch`